Universidad Carlos III de Madrid
www.uc3m.es

Trabajo Fin de Grado:

Sistema de ejecución de gestos y formas abstractas mediante interpolación spline en un robot social.

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

AUTOR:
Guillermo Pavón Gray

TUTOR:
Javier Fernández de Gorostiza Luengo

Leganés, 25 de Junio de 2013

Agradecimientos

Quisiera agradecer, en primer lugar, a Javier Gorostiza, que ha sido mi tutor durante todo este tiempo, ofreciéndome apoyo y que me acogió cuando estuve huérfano de proyecto.

También a todos los que, de forma permanente o de un modo más intermitente han estado trabajando en el laboratorio de Maggie al mismo tiempo que yo, por amenizar los ratos y ofrecerme ayuda cuando se la pedí.

A mi familia, a mi padre y a mi madre, porque gracias a ellos no me ha faltado de nada para poder atravesar esta etapa.

A mis compañeros del grado. Especialmente a Norber, Cris, Gabri, Bor y Jorge. Por hacer de mi estancia en la universidad un tiempo mucho más agradable, por su apoyo diario para que haya llegado hasta el final y pueda estar ahora mismo escribiendo estas páginas.

A los mineros y no tan mineros. A Borja, Fer, House, Lukas, Mil, Mersh y Rita. Por grandiosos momentos y risas infinitas, por sacarme de Madrid y de mi embotamiento cuando la cabeza y no me daba para más. Por entenderme cuando tenía que estudiar, y por obligarme cuando tenía que hacerlo, o dejar de hacerlo. Por todo el apoyo moral.

A los escritores (y la que no), Óscar, Luis, Mar, Luzmarina, Sofía y Laura. Por darme alas más allá de la ingeniería, por el intercambio de palabras, por los otros proyectos, por otro tipo de apoyo moral, por darme una dimensión distinta.

Resumen

Este trabajo comienza realizando una introducción a los robots sociales, y la comunicación no verbal, que servirá como motivación para el desarrollo completo. La idea general es conseguir que un robot social concreto, Maggie, de la Uc3m, sea capaz de realizar gestos para mejorar su comunicación con seres humanos. Y por extensión, que en un futuro lo acabe haciendo cualquier robot.

Existen dos vías de desarrollo que van a seguirse. Por un lado, se pretende realizar una interfaz para que el robot sea capaz de realizar un gesto a partir de un archivo con información de puntos que previamente ha interpolado mediante splines. El resultado será un movimiento coordinado de las articulaciones del robot representando una emoción.

Por otro lado, se pretende realizar una interfaz para que el robot sea capaz de, a partir de un mismo archivo con información de puntos, realizar el mismo movimiento con cualquiera de sus articulaciones. Como en el caso anterior, previamente también deberá haber interpolado esos puntos. Además, deberá procesar la información inicial para que el movimiento pueda ser ejecutado.

Para ello se comienza realizando una descripción exacta de la parte técnica, por una parte, de los conceptos que se van a introducir y un informe más detallado de los pasos a seguir. Por otra parte, de la plataforma de trabajo existente y los elementos que se van a utilizar para el desarrollo del trabajo.

Se continúa con una descripción exacta del funcionamiento del programa, las dos interfaces que realizarán los cometidos comentados arriba y sus funciones.

Para finalizar, se muestran los resultados experimentales, los posibles trabajos futuros y las conclusiones del trabajo.

Abstract.

This document starts with an introduction about social robots and non-verbal communication, which will serve as a motivation for the whole development. The general idea is to manage that a specific social robot, Maggie, from the Uc3m, can be capable of performing gestures to improve his communication with human beings.

There are two ways of development to be followed. On the one hand, is intended to create an interface so the robot can be capable of perform a gesture just with the information gathered from a file. This information must be previously interpolated by spline method. The result will be a coordinated movement of the robot joints that express an emotion.

On the other hand, is intended to create an interface so the robot can be capable of performing exactly the same movement with any one of the joints with the same single file for all of them. As in the previous case, before that the points of the gesture shall be interpolated. Also, the initial information of the file has to be processed in order to execute the movement.

To manage all the things mentioned before, this document starts with a description of the technical part. First, the new concepts that will be introduced, as well as a more detailed inform of the steps to follow. Secondly, a description of the existing work platform and the elements that will be used for the development.

It continues with an exact description of the developed program, the two interfaces that will achieve the specifications met before and its functions.

Finally, it shows the experimental results, and describes the possible future works and the conclusions gathered.

Índice

Índice de figuras

1.- Introducción

1.1.- Estructura del documento

1. Introducción. En este capítulo se introduce al trabajo y a su contenido mediante la estructura del mismo, la motivación para su realización y los objetivos principales.

2. Estado del arte. En este capítulo se enumeran y explican brevemente algunos robots comerciales cuya estructura humanoide puede recordar en cierto modo a Maggie.

3. Planificación. En este capítulo se exponen brevemente los pasos a seguir para realizar el trabajo y el tiempo estimado para conseguirlo.

4. Sistema de ejecución de gestos y formas abstractas mediante interpolación spline. En este capítulo se explica exactamente en qué consiste el proyecto, qué se va a hacer y cómo, además de la descripción de los elementos que en él se utilizan.

5. Plataforma de desarrollo. En este capítulo se explica sobre qué plataformas existentes se va a apoyar el presente trabajo. Esto incluye una descripción del hardware y el software utilizados, las interfaces de movimiento existentes, la interpolación spline, la biblioteca externa que se ha utilizado para interpolar y el uso de vectores.

6. Desarrollo e implementación. En este capítulo se detalla el funcionamiento exacto de las dos interfaces que se implementan, desgranando las funciones que componen cada una de ellas y explicando los problemas encontrados y cómo se solucionaron.

7. Resultados experimentales. En este capítulo se muestran los resultados obtenidos, tanto de forma cualitativa como de forma cuantitativa.

8. Presupuesto. En este capítulo se hace una estimación del coste total de la realización del proyecto.

9. Trabajos futuros. En este capítulo se exponen las posibles alternativas en las que la aplicación puede ser usada, y las posibles ampliaciones al programa.

10. Conclusiones. En este capítulo se detallan las conclusiones extraídas de la realización del trabajo.

Anexo I: Lista de los índices de articulación que se emplean en las variables, los ficheros de gesto o en la función para convertir una forma en un gesto.

Anexo II: Splinegesture.h. Declaración de la función para ejecutar un gesto.

Anexo III: Splinegesture.cpp. Función para ejecutar un gesto.

Anexo IV: Splineform.h. Declaración de la función para ejecutar un gesto a partir de una forma abstracta.

Anexo V: Splineform.cpp. Función para ejecutar un gesto a partir de una forma abstracta.

Anexo VI: Vídeo de la ejecución del gesto “Happy”.

Anexo VII: Vídeo de la ejecución del gesto “Sad”.

Anexo VIII: Vídeo de la ejecución del gesto “Angry”.

Anexo IX: Vídeo de la ejecución del gesto “Thinking”.

Anexo X: Vídeo de la ejecución del gesto a partir de una forma abstracta usando el brazo izquierdo.

Anexo XI: Vídeo de la ejecución del gesto a partir de una forma abstracta usando el movimiento vertical del cuello.

Anexo XII: Vídeo de la ejecución del gesto a partir de una forma abstracta usando el párpado izquiedo.
Anexo XIII: Vídeo de la ejecución del gesto a partir de una forma abstracta usando el movimiento horizontal del cuello.

Anexo XIV: Vídeo de la ejecución del gesto a partir de una forma abstracta usando el movimiento horizontal del cuello con un bias en la posición.

Anexo XV: Vídeo de la ejecución del gesto a partir de una forma abstracta usando el movimiento horizontal del cuello con un bias en el tiempo.

Anexo XVI: Vídeo de cambio de gesto, pasando de "Sad" a "Happy" durante la ejecución.

1.2.- Motivaciones para el trabajo. Lenguaje no verbal y empatía.

Una de las características más importantes y definitorias de la inteligencia es la capacidad de comunicación. La propia historia de la humanidad gira en gran medida en torno a la posibilidad de que un ser humano sea capaz de transmitir una idea o un conocimiento a otro ser humano.

La cultura, el desarrollo, la ciencia... todo esto sería imposible sin comunicación. Sin embargo, no es necesario tratar con ejemplos tan amplios. De forma mucho más cercana, a pie de calle, encontramos que la comunicación es el factor determinante a la hora de definir las relaciones humanas, o incluso entre animales, y se convierte en una pieza clave en factores tan importantes como la empatía.

El psicólogo Albert Mehrabian [1] estima que, para ciertas situaciones, el porcentaje de la comunicación que está directamente ligado con el lenguaje corporal es aproximadamente de un 55%. Si bien esto depende mucho del contexto comunicativo, sí que nos da una idea del gran peso que tiene el lenguaje corporal en la comunicación.

Los gestos, la respiración, los tics, cualquier acción de nuestro cuerpo, hasta el movimiento más insignificante, esconde una dosis de comunicación, un mensaje visual al fin y al cabo, que podrá ser, y de hecho generalmente será, interpretado por el interlocutor.

El mundo de la robótica avanza cada vez de forma más rápida, y con él, el de los robots sociales. Para garantizar su integración, el lenguaje no verbal es una herramienta muy poderosa, es una barrera clave a la hora de plantearse una interacción efectiva entre el mundo humano y el robótico. De ahí la motivación de disponer de una interfaz capaz de ejecutar de forma sencilla distintos gestos en diferentes robots.

Hemos hablado antes de la empatía, la capacidad de que dos individuos sean capaces de sentir lo mismo, o de forma parecida en un contexto. Probablemente la empatía es uno de los pilares de las relaciones entre los individuos. La posibilidad de sentir amor o afecto está poderosamente ligada a la empatía.

Los seres humanos podemos sentir empatía fácilmente con otros seres humanos, pero también nos resulta enormemente sencillo hacerlo con animales. Y si es posible que sintamos empatía por otros seres vivos, es cuestión de tiempo que, aunque sin llegar a los mismos niveles, que sintamos un mayor y más poderoso vínculo con un robot. La comunicación no verbal tiene un importante papel en todo esto; es un paso más en el camino a que los robots sean vistos menos como simples máquinas y más como compañeros.

1.3.- Objetivos.

El objetivo principal del presente trabajo es desarrollar el uso de gestos en un robot social para aumentar su expresividad no verbal. Para ello, se han propuesto dos objetivos más concretos:

- Diseño e implementación de una interfaz de ejecución de gestos para el robot social Maggie basada en la interpolación spline.

- Diseño e implementación de una interfaz de ejecución de gestos para robots basada en la interpolación spline a partir de formas abstractas.

La primera interfaz debe ser capaz de leer un archivo de texto con un gesto normalizado, extraer los puntos del gesto de él, realizar una interpolación mediante splines en los puntos extraídos, separar esa curva en pequeños intervalos, interpretar esos intervalos como posiciones del robot y convertirlos en un movimiento coordinado de las articulaciones de Maggie.

La segunda interfaz debe ser capaz de, utilizando lo anterior como base, leer ficheros de texto con una forma abstracta, normalizar la forma y reescalarla en posición y en tiempo, convertirla en un gesto asociándola a una articulación cualquiera, realizar una interpolación mediante splines en los puntos de la forma, separar esa curva en pequeños intervalos, interpretar esos intervalos como posiciones del robot y convertirlos en un movimiento coordinado de las articulaciones de Maggie.

Como objetivos secundarios del proyecto se plantean dos:

- Por un lado, crear un pequeño conjunto de gestos de muestra para que puedan ser interpretados por la interfaz ejecutora de gestos. Estos gestos deberán expresar alguna emoción del robot reconocible por parte de cualquier ser humano. Además, puesto que son pruebas, deberán mover el máximo número posible de articulaciones.

- Por otro lado, como ampliación al programa, implementar la posibilidad de que pueda cambiarse el gesto mientras se está ejecutando.

2.- Otros robots humanoides.

Hablamos de robots sociales para referirnos principalmente a robots no industriales. Es decir, robots autónomos creados para el beneficio y bienestar de los seres humanos y capaces de interactuar con ellos.

Un robot social debe interactuar con las reglas sociales relacionadas con su rol. Este rol y las reglas por las cuales actúa el robot estarán definidos por su entorno y la sociedad.

Aunque es posible la interacción de robots con humanos reduciendo al mínimo la comunicación entre ellos, o incluso eliminándola (por ejemplo, el robot de limpieza Roomba prácticamente la elimina) lo habitual es que exista cierta comunicación para que el robot sea considerado social.

Una parte fundamental de la comunicación es la no verbal. A continuación se exponen algunos de los robots actuales de morfología humanoide similar a la de Maggie, y por tanto capaces de realizar gestos de forma parecida. La mayoría de ellos, al ser robots comerciales, no tienen mucha información accesible.

NAO. [2] [3] Es un robot humanoide desarrollado por Aldebaran Robotics. El robot comenzó a desarrollarse en 2004. Es el robot oficial utilizado en la Robocup Standard Platform League, una competición internacional de fútbol con robots como jugadores. Se ha utilizado con diversos fines académicos y de aprendizaje relacionados con la robótica.

Puesto que uno de sus fines es académico, resulta ser de los pocos robots comerciales de los que se puede encontrar cierta información sobre su arquitectura.

Existen varias versiones del robot Nao, que le confieren 25, 21 o 14 grados de libertad. A continuación se muestra una imagen con los grados de libertad del robot, además de algunos de sus sensores y actuadores. Debajo, una tabla con las articulaciones del robot en su modelo de 25 grados de libertad.

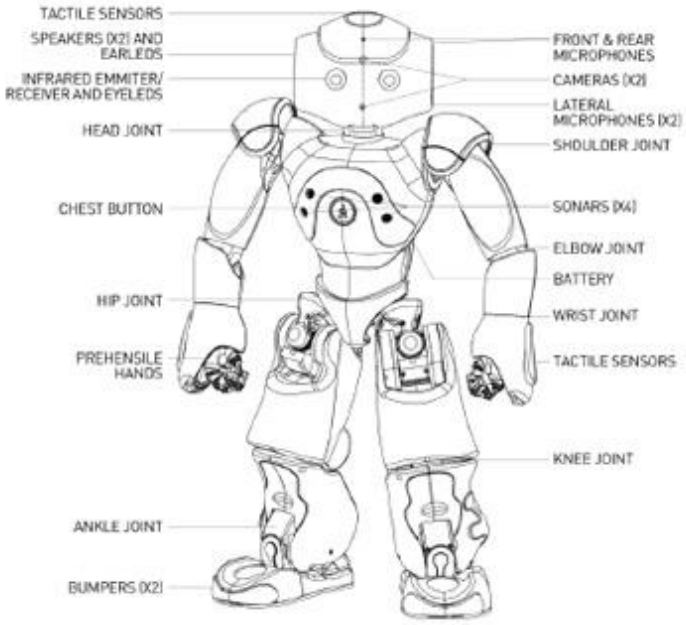


Figura 1: Articulaciones del robot NAO

Head + LArm + LLeg + RLeg + RArm				
Head	**LArm**	**LLeg**	**RLeg**	**RArm**
HeadYaw	LShoulderPitch	LHipYawPitch[1]	RHipYawPitch[1]	RShoulderPitch
HeadPitch	LShoulderRoll	LHipRoll	RHipRoll	RShoulderRoll
	LElbowYaw	LHipPitch	RHipPitch	RElbowYaw
	LElbowRoll	LKneePitch	RKneePitch	RElbowRoll
	LWristYaw[2]	LAnklePitch	RAnklePitch	RWristYaw[2]
	LHand[2]	RAnkleRoll	LAnkleRoll	RHand[2]

Figura 2: Tabla de articulaciones de NAO

La interfaz de movimiento de Nao tiene tres opciones. El control básico puede mover cualquier articulación a una posición determinada a una velocidad determinada. Tiene además un control mediante interpolación de puntos, por si se conoce de antemano la trayectoria a seguir, y un movimiento para usar con control reactivo.

2. Otros robots humanoides

HOAP. [4] Es un robot desarrollado por la empresa Fujutsu en 2001. La última versión, HOAP 3 salió en 2005.

HOAP posee 28 grados de libertad. Seis de ellos en las piernas, cinco en cada brazo, uno en cada mano, uno en la cintura y tres en el cuello.

Figura 3: Robot HOAP

ASIMO. [5] [6] Es un robot diseñado por Honda. Se introdujo en el año 2000, con la intención de que fuese un robot asistente para ayudar a personas con movilidad reducida. Es probablemente uno de los robots humanoides más famosos, y ha aparecido en multitud de eventos y demostraciones relacionados con la robótica, las ciencias y las matemáticas.

ASIMO fue el primer robot en introducir un control de movimiento predictivo, lo que le confiere mayor flexibilidad en las articulaciones y un movimiento más suave y parecido al de los seres humanos.

2. Otros robots humanoides

A partir de 2007, ASIMO es capaz de interactuar con otros robots para trabajar juntos de forma coordinada.

ASIMO tiene en total 34 grados de libertad. Cuello, hombro, muñeca y cadera tienen tres grados de libertad cada uno. Cada mano tiene un pulgar articulado con dos grados de libertad. Cada tobillo tiene dos grados de libertad, y cintura, rodillas y hombros tienen un grado de libertad cada uno.

Figura 4: Robot ASIMO

QURIO. [7] Es un robot desarrollado por Sony que no llegó a comercializarse. Nace con la intención de ser una continuación en forma de robot humanoide con AIBO, un robot con forma de perro creado y comercializado con fines de entretenimiento. En enero de 2006, Sony informó que cesaría el desarrollo tanto de AIBO como de QRIO.

QRIO fue el primer robot bípedo capaz de correr (es decir, levantar ambas piernas del suelo al mismo tiempo). El robot QRIO tiene 38 grados de libertad.

Figura 5: Robot QRIO.

Los robots arriba mencionados se caracterizan por tener una gran movilidad en su estructura, pero sus caras son estáticas, carecen de movimiento, y por tanto son incapaces de mostrar ninguna emoción. Ni siquiera como Maggie, que al menos posee la capacidad de mover los párpados. Esto es debido a que la expresividad facial es tremendamente compleja, y por lo general, los robots comerciales no suelen explotar esta capacidad. Son robots más específicos aquellos en los que se pretende desarrollar este tipo de facultades, y están orientados casi exclusivamente a este propósito. A continuación se muestran, a modo de ejemplo, dos robots centrados en la expresividad facial.

Kismet y Leonardo. Kismet [8] [9] es una cabeza antropomórfica desarrollada por el MIT. Cuenta con 21 grados de libertad que le permiten mover ojos, párpados, cejas, cuello, labios y orejas. Está inspirado en el modo en el que los niños interactúan con adultos.

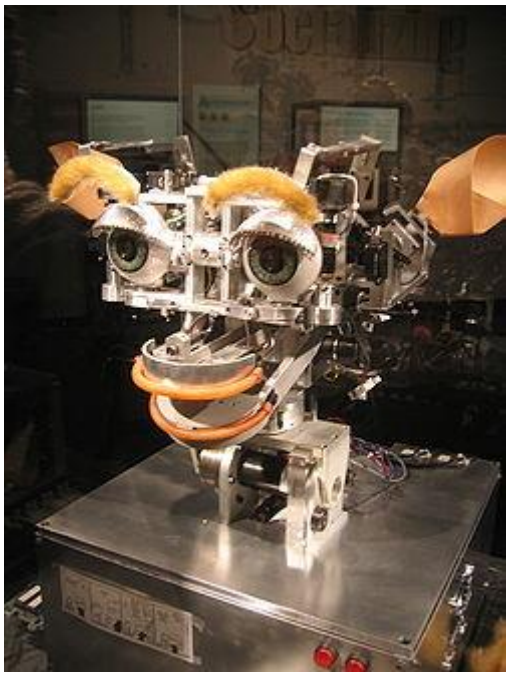


Figura 6: Robot Kismet

Leonardo, [9] [10] desarrollado por la misma universidad, es un robot pensado para la interacción con humanos. Está recubierto en piel de peluche, lo que lo hace que su interacción con personas resulte más agradable. Cuenta con aproximadamente sesenta motores para ayudar a que su cara y cuerpo sean lo más expresivos posibles.

Figura 7: Robot Leonardo

Ambos robots basan su arquitectura interna en un sistema reactivo, es decir, están preparados para responder a los estímulos que reciben y al comportamiento de la persona que tienen delante. Por el contrario, carecen de capacidad de aprendizaje.

3.- Planificación.

3.1.- Primera fase. Preparación.

En esta primera fase se acometen las tareas previas al desarrollo de las APIs.

- Investigación del estado del arte.

- Conocimiento del robot Maggie. Antes de comenzar a desarrollar sobre Maggie, es necesario conocer el robot, sus particularidades, sus articulaciones y las posibilidades que estas presentan, si existen APIs o programas antiguos que puedan servir y la arquitectura sobre la que funciona.

- Realizar tutoriales de la arquitectura ROS. Al finalizar el desarrollo del programa, se adaptará para que sea utilizado sobre la arquitectura ROS, por tanto es importante empezar a familiarizarse con ella.

- Buscar información sobre la interpolación spline, y buscar bibliotecas compatibles con Linux y C++ que sean capaces de generar curvas de interpolación.

3.2.- Segunda fase. Inicio de desarrollo.

En esta fase se busca aplicar los conocimientos adquiridos en la fase anterior y comenzar a desarrollar la primera de las APIs

- Prueba de movimiento individual: creación de un programa de prueba con el que mover cada uno de los seis grados de libertad a las posiciones deseadas.

- Prueba interpolación spline: creación de un programa de prueba con el que generar curvas splines a partir de los datos de un vector de posición y uno de tiempo.

- Prueba movimiento spline: creación de un programa de prueba que permita mover una articulación usando puntos de una interpolación spline.

- Prueba movimiento conjunto: creación de un programa de prueba que permita mover al mismo tiempo todos los grados de libertad del robot con puntos extraídos de una curva spline.

3.3.- Tercera fase. Desarrollo.

- Desarrollo de las funciones de la clase *splinegesture.* En la medida de lo posible, utilizar código que haya funcionado en las pruebas anteriores.

- Prueba *splinegesture*: creación de un programa de prueba para verificar el funcionamiento de la clase *splinegesture*. Desarrollo de algunos gestos para comprobar que el movimiento deseado se ajusta a la realidad. Corrección de errores.

- Desarrollo de las funciones de la clase *splineform*. En la medida de lo posible, utilizar código que haya funcionado en la clase *splinegesture.*

- Prueba *splineform*: creación de un programa de prueba para verificar el funcionamiento de la clase *splineform*. Pruebas en ambas clases.

3.4.- Cuarta fase. Ampliación, y finalización del código.

- Desarrollo de la ampliación: posibilidad de modificar el gesto mientras este se está ejecutando.

- Prueba ampliación: creación de un programa de prueba para verificar que la ampliación funciona correctamente.

- Prueba global de que todo el programa funciona a la perfección.

- Optimización del código, borrado de todos aquellos elementos que solo servían para la realización de pruebas. Comentar el código.

- Desarrollo de la memoria del proyecto.

3.5.-Diagrama de Gantt.

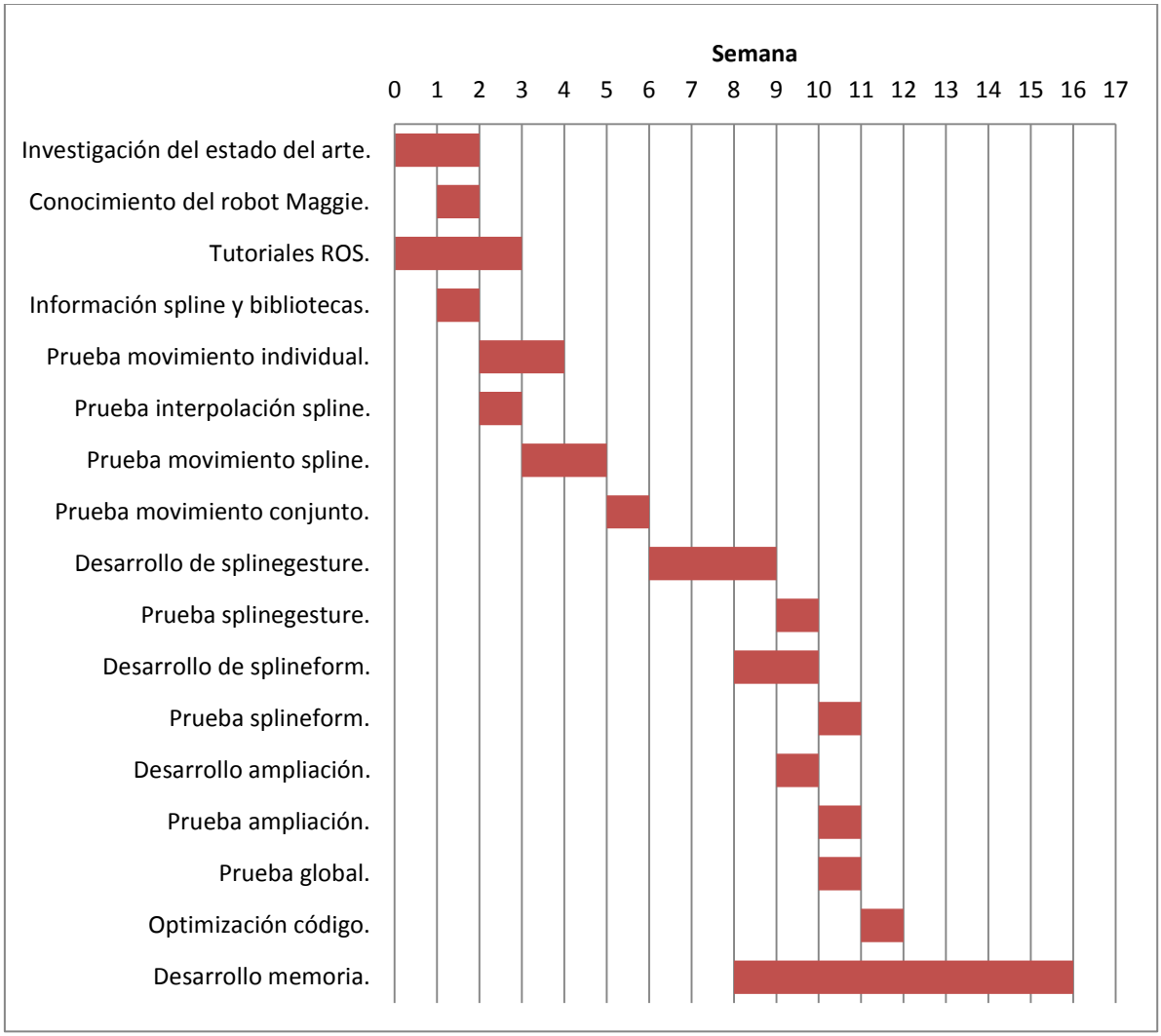


Figura 8: Diagrama de Gantt de la planificación.

4- Sistema de ejecución de gestos y formas abstractas mediante interpolación spline.

4.1.- Gestos y formas abstractas.

En el lenguaje formal se considera gesto a un acto intencionado de comunicación no verbal ejecutado con alguna parte del cuerpo y producida por el movimiento de las articulaciones a través de los músculos.

Extrapolando de lo anterior, un gesto en el robot social Maggie será también un acto intencionado de comunicación no verbal. Y su ejecución será en forma de movimiento de una o varias de sus articulaciones. Acciones sencillas como asentir, guiñar un ojo o señalar son consideradas gestos; acciones más complejas como mover los brazos en posición de enfado o expresar alegría también son consideradas gestos. Sin embargo, acciones que no conllevan intención comunicativa no son consideradas gestos, por ejemplo, girarse para orientar la cámara no es un gesto.

Por extensión, también se hablará de gestos para referirse a un conjunto de puntos con instrucciones precisas para que al ejecutarse creen un gesto único. Es decir, será considerado gesto el conjunto de puntos susceptible de ser ejecutado sin modificación alguna, siempre y cuando esta ejecución sea también considerada un gesto. En el presente trabajo, consistirán en conjuntos de pares de puntos posición-tiempo. Cada uno de estos pares de puntos estará asociado a una articulación concreta. Por ejemplo, un conjunto de puntos que contenga las instrucciones para mover el cuello arriba y abajo asintiendo, es ya considerado un gesto incluso antes de ejecutarse.

Una forma abstracta es una función de valores reales donde los valores del eje X, que deben ser siempre superiores a 0, representan el tiempo y los valores del eje Y representan la posición. Una forma abstracta representa un movimiento, pero no indica qué articulación es la que debe realizar dicho movimiento, por tanto tampoco indica posiciones absolutas (no pueden indicarse posiciones absolutas referenciadas a una articulación que se desconoce). Tanto las posiciones como los tiempos de una forma están expresadas en valores relativos unos a otros, como un porcentaje del

máximo valor de la forma. La información que almacena una forma, son proporciones: qué amplitud tiene una posición de la forma con respecto a las demás, y cómo de veloz debe ejecutarse un movimiento con respecto a los otros.

En la figura, se muestra un ejemplo de posible forma abstracta. Esta forma ejecutaría al principio un movimiento rápido y amplio y a continuación uno más lento y menos amplio.

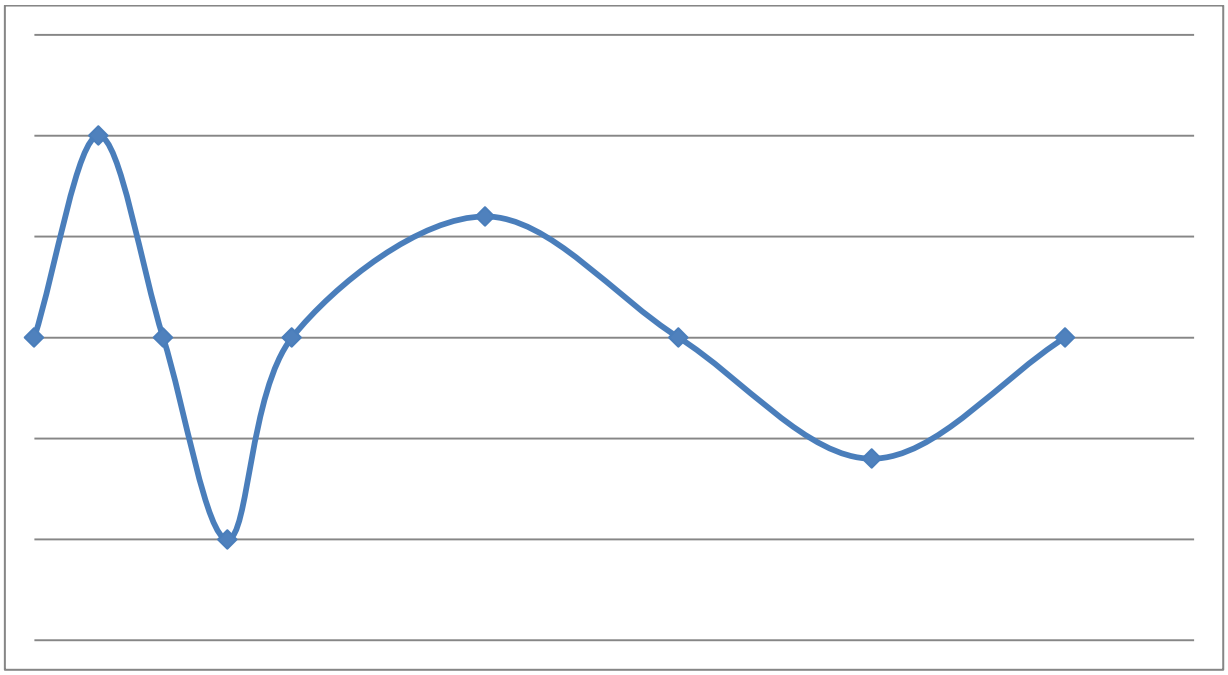


Figura 9: Ejemplo de forma abstracta.

Antes de convertirse en movimiento, una forma abstracta debe procesarse. El procesado incluye normalización, asociación a una articulación y escalado con respecto a las posiciones absolutas de esa articulación, y un posible reescalado en el que se varían tanto la amplitud como la duración globales al multiplicarlos por un bias. Tras el procesado, deja de ser una forma abstracta para ser un gesto.

Los gestos tienen la ventaja de que están completamente definidos y pueden ser ejecutados instantáneamente. Las formas, por otra parte, son más versátiles, ya que pueden ser ejecutadas por cualquiera de las articulaciones del robot, incluso podrían ser ejecutadas por otros robots, pero necesitan ser procesadas.

En el lenguaje más básico, un gesto sería como decir “Inclina el brazo derecho treinta grados hacia arriba durante dos segundos y después vuélvelo a bajar”, mientras que una forma sería algo similar a “Mueve a la posición 30 en dos segundos, y después mueve a la posición 0”. La segunda proposición no puede ejecutarse, porque no está definido ni qué parte del cuerpo debe moverse ni cuáles son las posiciones 30 y 0.

4.2.- Diseño de la solución.

Se van a desarrollar dos programas independientes. Uno de ellos se encargará de tomar datos de un gesto a partir de un fichero .xml, interpretar esos datos como posiciones para sus distintas articulaciones, calcular una curva de interpolación spline sobre esos puntos y ejecutar el gesto convirtiendo esa curva en un movimiento coordinado de las articulaciones del robot.

El otro deberá leer ficheros .xml que describan una forma abstracta, normalizarlos, asociarlos a una articulación del robot por determinar, calcular una curva de interpolación spline sobre ellos y convertir esa curva en movimiento de dicha articulación.

Se sabe que es probable que ambos programas compartan cierta cantidad de código, por lo que tal vez sería más eficiente juntarlo todo en uno solo. Sin embargo, cada uno de los programas va a ser usado para una aplicación distinta. Sin ir más lejos, es posible que la clase encargada de convertir un archivo de forma abstracta en un gesto pueda ser utilizado en robots distintos a Maggie, mientras que la que lee gestos específicos es exclusiva de Maggie. Por tanto, se considera que la creación de dos programas independientes es una solución mucho más limpia, además de tener mayor vista hacia el futuro.

A ambos programas se les va a enviar una serie de pares puntos-posición a través de un fichero, que deben ser interpretados y ejecutados como gestos. La primera función de ambos es extraer esos puntos para poder trabajar con ellos.

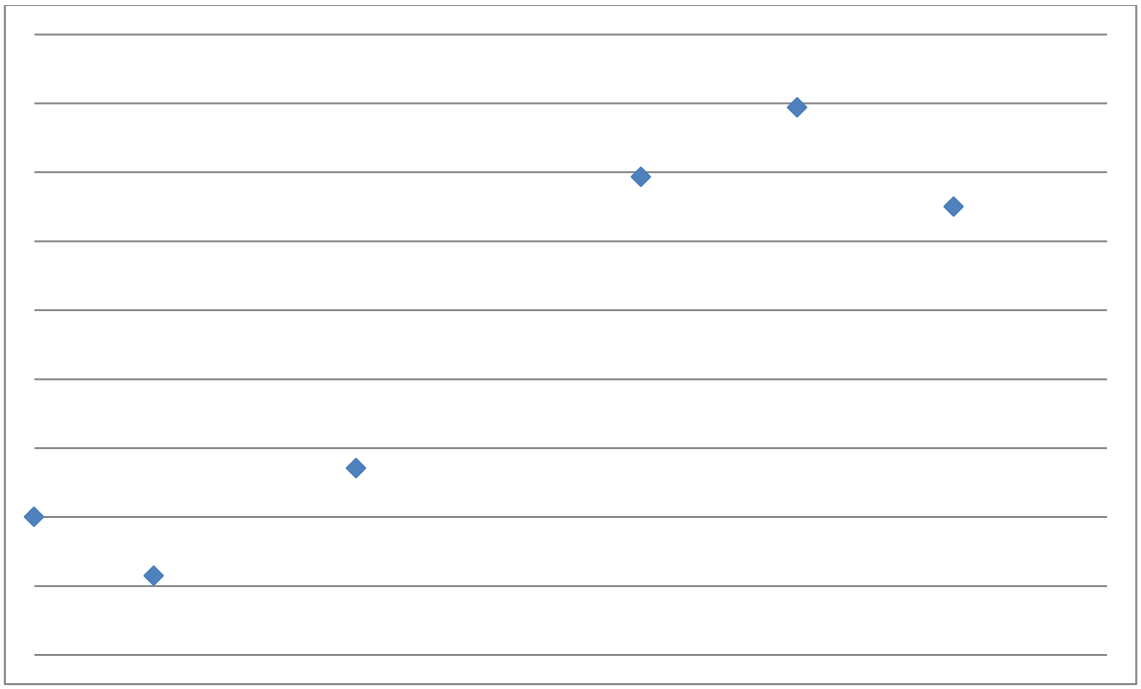
Se ha adelantado que las formas abstractas necesitan ser procesadas. Por tanto, en este punto del programa que trabaja las formas abstractas será necesario hacer dicho procesado. En primer lugar normalizar la forma a unos estándares determinados. De

este modo las siguientes funciones serán válidas sea cual sea la forma abstracta que se introduzca en el programa. En segundo lugar, de forma opcional, se podrá escalar la forma normalizada para cambiar su amplitud o su duración temporal. En ningún momento el proceso de escalado afecta a la proporción de la forma, que es, como se ha mencionado, la información real que la forma abstracta tiene. El último apartado del procesado de formas consiste en asociar esa forma a una articulación, y convertir los valores relativos de la forma normalizada en valores absolutos de la articulación. En este momento, la forma deja de serlo para convertirse en un gesto. Este procesado no es necesario en el caso de gestos ni está, por tanto, implementado en el programa que trabaja con ellos.

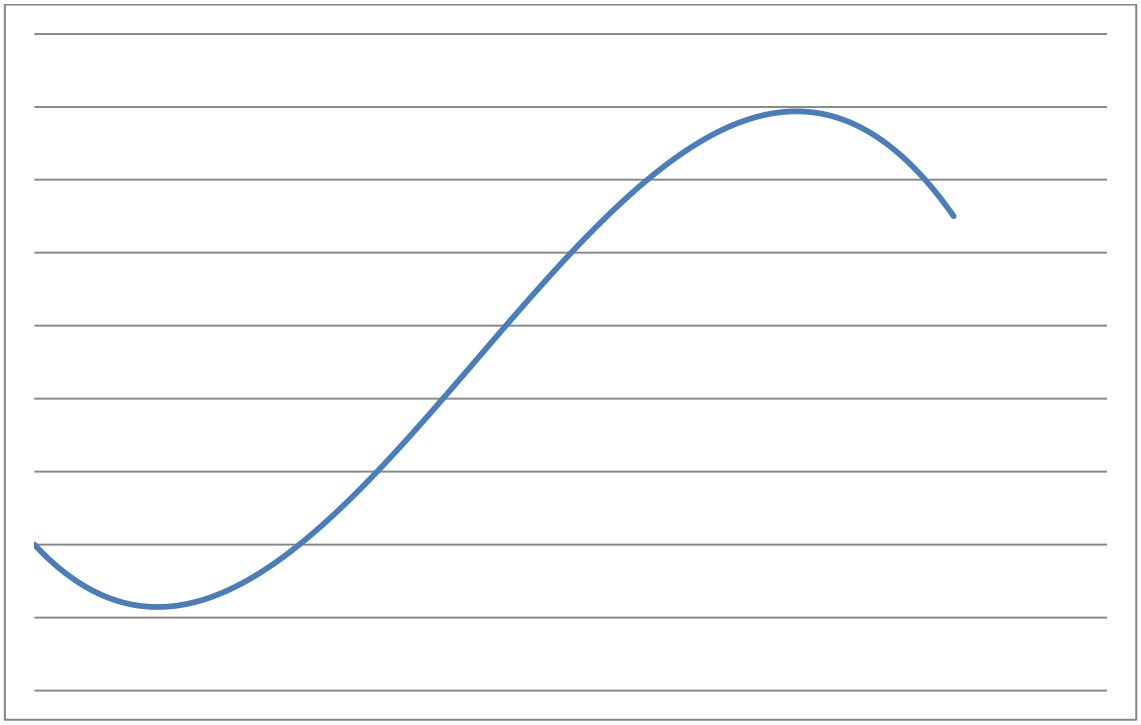
Para generar un movimiento constante, regular y controlado en todo momento, se va a utilizar la interpolación spline para generar una curva de movimiento que seguirá el robot entre los puntos que se extraigan del fichero. A partir de la curva se va a dividir el movimiento del robot en intervalos de 30 milisegundos. En cada uno de esos intervalos se va a mover el robot a una posición.

Es necesario un tiempo de refresco a la hora de copiar datos por el puerto serie de los servos que efectúan el movimiento de Maggie. El tiempo de refresco teórico es de 70 milisegundos. Inicialmente el intervalo que se escogió para extraer puntos de la interpolación spline fue de esos 70 milisegundos. Sin embargo, en las pruebas se descubre que es un intervalo demasiado grande, y produce un movimiento ligeramente discontinuo. Al reducir ese intervalo a 30 milisegundos, el movimiento vuelve a ser continuo y natural.

Puntos iniciales extraídos del fichero:



Interpolación spline de los puntos:



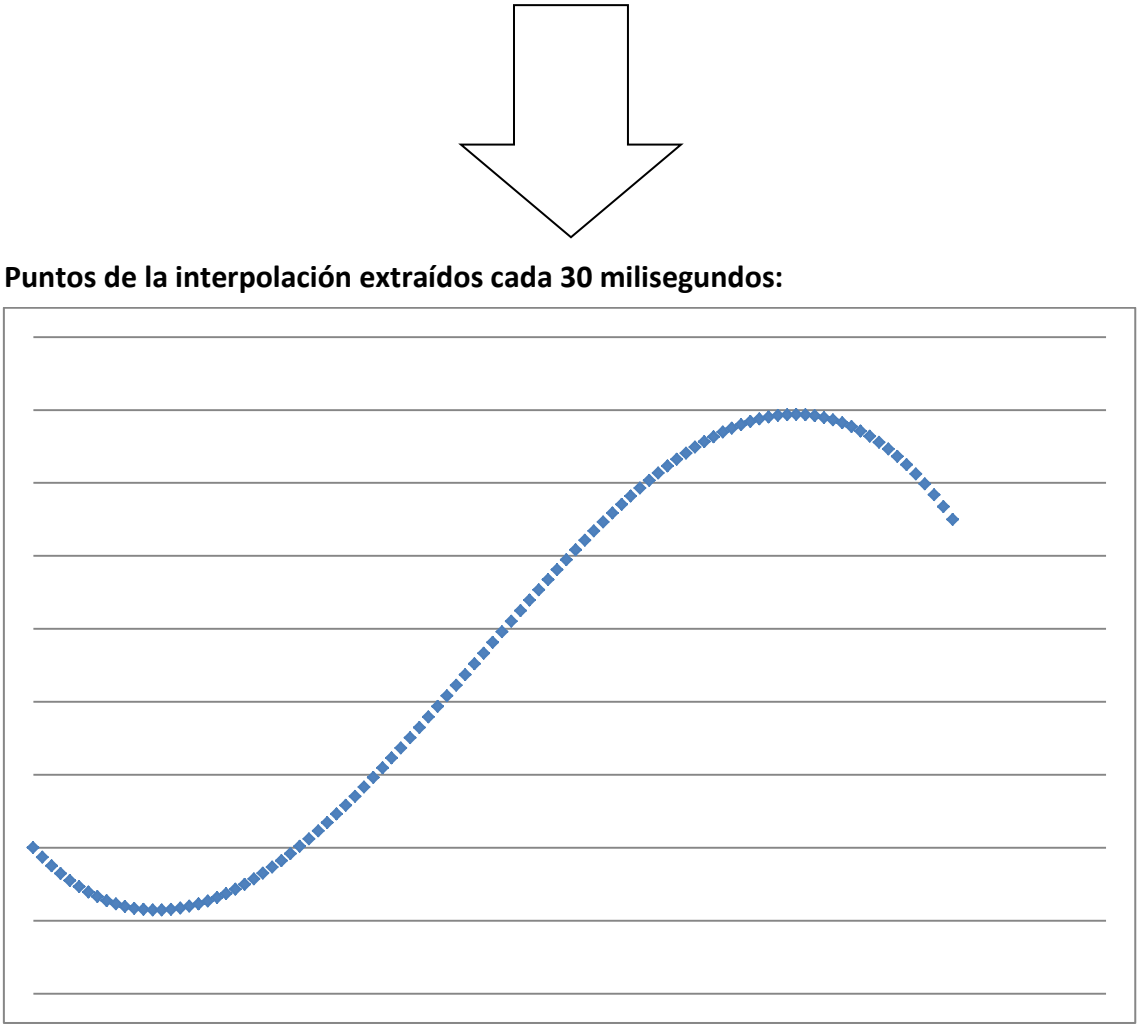


Figura 10: Puntos de un fichero, interpolación y extracción de los puntos de la spline cada 30 milisegundos.

Es necesario recalcar que el único dato que va a enviarse a los servos del robot es la posición, pero no la velocidad. Sin embargo, puesto que se hace un control de posición y de tiempo, podemos hablar de un control indirecto de la velocidad de las articulaciones del robot.

Las ventajas que tiene hacer un control punto a punto de la posición son, en primer lugar, que se tiene conocimiento y control sobre el movimiento a lo largo de toda la curva, y en segundo lugar, puesto que cada 30 milisegundos estamos forzando a la articulación a moverse a un punto, se corrigen los posibles errores de posición a cada punto. Las ventajas de un control punto a punto sobre la velocidad son, además de que, como en la posición, se tiene control sobre ella durante toda la curva y se corrigen

los posibles errores cada vez que se da la orden de moverse a un nuevo punto, también es posible cambiar el valor de la velocidad durante la curva.

Una vez que se tienen las posiciones del gesto en forma de puntos extraídos de la interpolación, y asociados a una articulación, se enviarán los puntos al servo de la articulación para que se efectúe el movimiento. Puesto que los puntos están separados en un intervalo de 30 milisegundos, cada vez que se envíe un punto será necesario esperar ese espacio de tiempo antes de poder enviar el siguiente a la misma articulación. SI se desean mover varias articulaciones al mismo tiempo, será necesario enviar en el mismo intervalo de 30 milisegundos el siguiente punto a moverse para cada una de las articulaciones que realizan movimiento.

5.- Plataforma de desarrollo

5.1.- El robot social Maggie

5.1.1.- Hardware de Maggie.

El hardware es la parte física. Su importancia en cualquier robot es más que obvia, sin embargo, cobra aún más importancia cuando hablamos de un robot social, y más aún a la hora de querer trabajar con gestos en dicho robot.

La arquitectura hardware de Maggie [11] se basa en una serie de sensores y actuadores gobernados por un ordenador interno y recubiertos por una carcasa de fibra de vidrio. Entre los sensores encontramos sensores capacitivos de tacto, distribuidos bajo la carcasa; un telémetro laser en la base, sensores SONAR, sensores infrarojo, una cámara web o un micrófono inalámbrico.

Figura 11: Robot Maggie.

Cuenta, como se ha adelantado, con un ordenador interno, llamado Vissmaggie, que es el que controla todas las operaciones internas del robot. Tiene además una conexión wireless, que le permite tanto conectarse a internet como permitir la posibilidad de que un ordenador externo pueda acceder a Vissmaggie. Además de Vissmaggie, en la parte frontal tiene un Tablet PC para reproducción de imágenes, vídeos o incluso juegos.

En cuanto a actuadores, cuenta, por ejemplo, con altavoces stereo y ruedas en la base que le permiten desplazarse por el entorno. Sin embargo, de cara a este proyecto y a la ejecución de gestos, nos centraremos en los actuadores que van a ser utilizados para la ejecución de dichos gestos, es decir, los de los brazos, cuello y párpados.

- Brazos: Cada uno de los brazos tiene un diseño sencillo, con un único grado de libertad, que le permite girar desde el hombro en un eje perpendicular al tronco. Cada brazo se controla con un motor de corriente continua y un encoder. El movimiento de cada brazo es independiente y pueden moverse por separado. El giro se transmite desde el motor a una polea mediante una correa de distribución, el movimiento se transmite al brazo con un eje que gira solidario con la polea y el propio brazo.

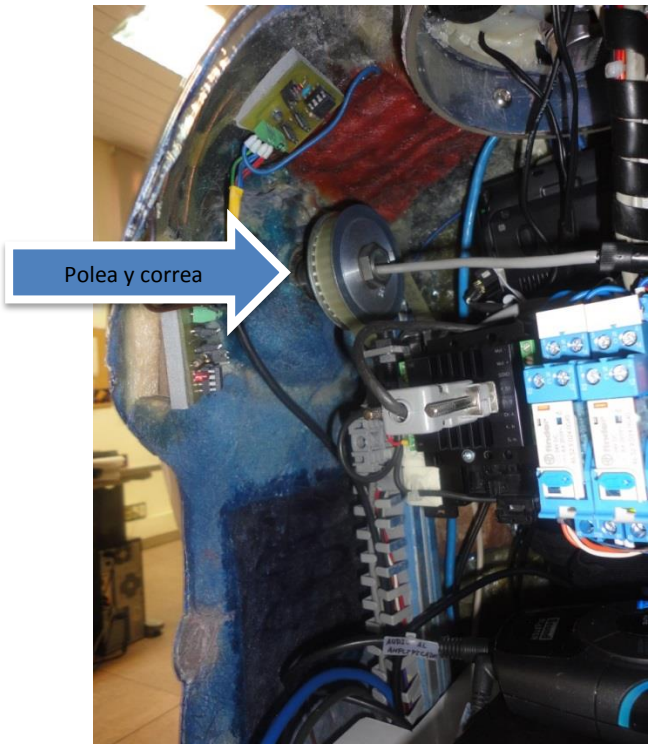


Figura 12: Polea y correa de distribución del brazo izquierdo montados.

- Cuello: El cuello es responsable del movimiento de toda la cabeza. Tiene dos grados de libertad, uno para el movimiento vertical, o theta, y otro para el movimiento horizontal o phi. Está formado por una estructura de aluminio que mueve la carcasa, pero deja estáticos el resto de componentes de la cabeza.

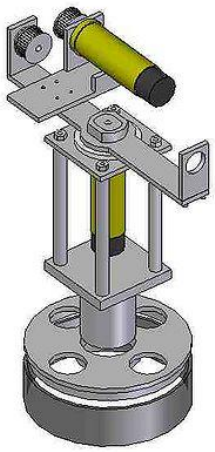


Figura 13: Estructura del cuello de Maggie.

Cada uno de los dos grados de libertad del cuello está controlado por un motor de corriente continua y un encoder.

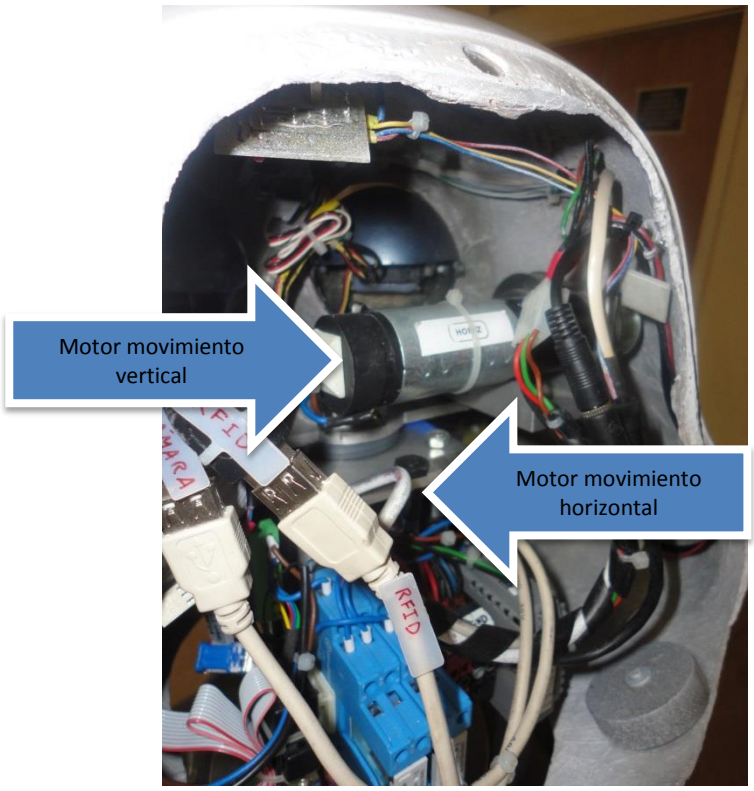


Figura 14: Estructura del cuello de Maggie montada.

- Párpados: Los párpados de Maggie están formados por dos semiesferas que giran cubriendo o descubriendo los ojos, simulando un parpadeo, o un guiño, y añadiendo expresividad al robot.

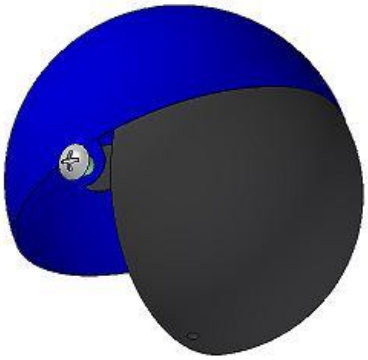


Figura 15: Estructura de los párpados de Maggie. El párpado(azul), al girar, cubre o descubre el ojo (negro).

Los párpados se controlan mediante servomotores, uno para cada ojo, y una placa de control capaz de controlar hasta ocho servomotores.

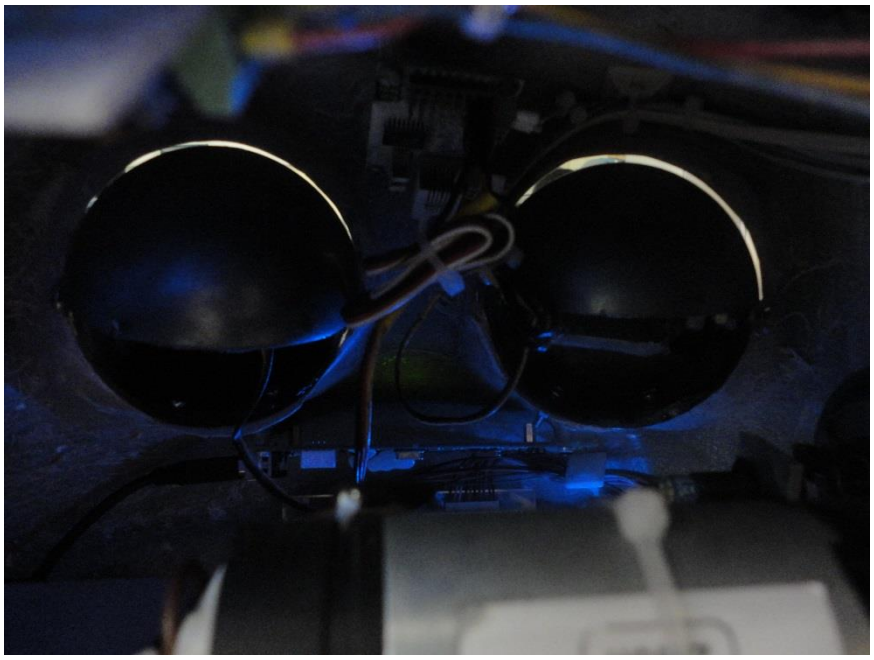


Figura 16: Ojos de Maggie montados vistos desde el interior.

5.1.2 Software de Maggie. Arquitectura AD.

Tradicionalmente, en el mundo de la robótica, se hablaba de dos posibilidades: arquitecturas planificadas o arquitecturas reactivas. Las primeras son buenas para la consecución de objetivos globales en entornos muy bien definidos, pero tienen problemas al encontrarse con situaciones inesperadas, o en entornos cambiantes. Las segundas reaccionan al entorno, pero no están pensadas para lograr grandes objetivos globales. Durante la década de los 90, ambas posibilidades empezaron a mezclarse en lo que se llamaron arquitecturas híbridas. [11]

A la hora de diseñar un robot social, se trata de emular o aproximarse al comportamiento de un ser humano. Un ser humano es capaz de planificar globalmente y al mismo tiempo reaccionar a su entorno, es decir, encajaría en el modelo de arquitectura híbrida.

La arquitectura de control AD, Automática/Deliberativa es una arquitectura de control híbrida y es la que controla a Maggie. Se encuentra descrita en la Tesis del Dr. Ramón I. Barber Castaño: "Desarrollo de una arquitectura para robots móviles autónomos. Aplicación a un sistema de navegación topológica". [12] Se divide en nivel deliberativo y nivel automático.

El nivel deliberativo es el encargado de todas aquellas actividades que requieran razonamiento previo y aprendizaje. Se compone de: habilidades deliberativas, que son aquellas relacionadas con el aprendizaje y el razonamiento, y deben procesarse de una en una, sin concurrencia; memoria a largo plazo, que contiene la información permanente o duradera proveniente de las habilidades deliberativas, y que usan después las propias habilidades deliberativas para realizar razonamiento; y un secuenciador que gestiona las habilidades deliberativas, seleccionando cuál se ejecuta en cada momento y definiendo, por tanto el comportamiento del robot.

El nivel automático controla los movimientos y sensaciones que no necesitan razonamiento ni se nutren de un aprendizaje. Estas habilidades pueden ser activadas por el nivel deliberativo o las acciones reflejas, o bien pueden ejecutarse de forma continua. En este nivel se encuentran los módulos de control a bajo nivel y los que recogen información directamente de los sensores. Está compuesto por un lado de las

habilidades automáticas, que son las capacidades de tacto sensoriales y las motoras del sistema, pueden ejecutarse de forma concurrente con otras, se activan en el nivel deliberativo, y la información que recogen se envía de nuevo al nivel deliberativo; y por otro lado de las acciones reflejas, que son la respuesta automática a un estímulo sensorial, y tienen carácter prioritario al resto de habilidades en el comportamiento del robot.

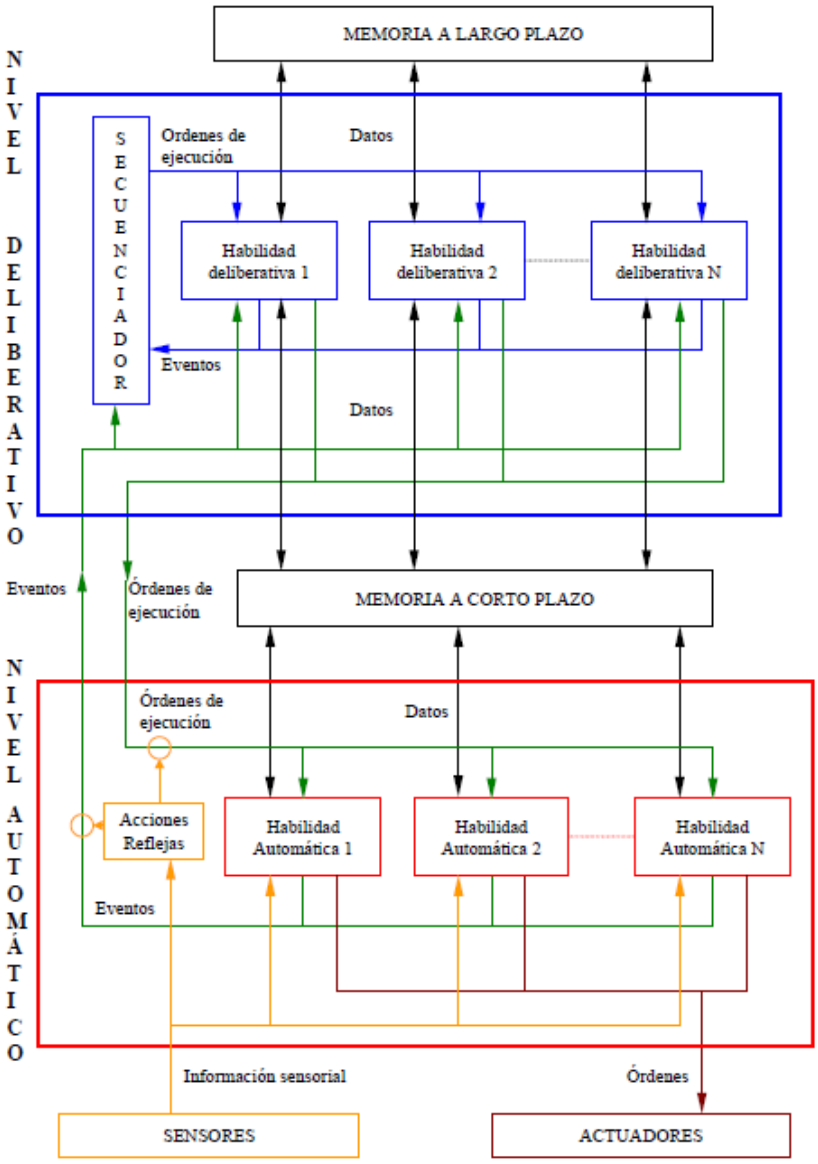


Figura 17: Esquema de funcionamiento de la arquitectura AD.

5.2 Interfaz de movimiento.

Existe una interfaz que controla el movimiento de las distintas articulaciones del robot, que utilizaremos en el presente trabajo para ejecutar los gestos. La interfaz consiste en una serie de clases C++, cada una con distintas funciones miembro para controlar las articulaciones. Existe una clase por cada tipo de articulación, por tanto habrá tres en total: una para ambos brazos, una para los dos movimientos del cuello y una tercera para los párpados.

Los brazos se gobiernan utilizando la clase *CarmBase*. Un objeto de esta clase puede servir para controlar cualquiera de los dos brazos, o los dos a la vez. Las funciones de esta clase que usaremos son:

- Función para inicializar los brazos, y permitir que puedan moverse.
- Función para mover brazo derecho a una posición.
- Función para mover brazo izquierdo a una posición
- Función para leer la posición actual del brazo izquierdo.
- Función para leer la posición actual del brazo derecho.

Esta clase C++ contiene además algunas funciones adicionales, como mover uno de los dos brazos a una posición con una velocidad determinada o mover ambos brazos a la vez a la misma posición, que no serán utilizadas en este trabajo.

El cuello se gobierna mediante la clase *NeckApi*. La clase construye un objeto que puede mover cualquiera de los dos grados de libertad, o los dos a la vez. Las funciones de esta clase que usaremos son:

- Función para inicializar el cuello, y permitir que pueda moverse.
- Función para mover el cuello en vertical a una posición.
- Función para mover el cuello en horizontal a una posición
- Función para leer la posición actual del cuello en vertical.
- Función para leer la posición actual del cuello en horizontal.

Esta clase C++ contiene algunas funciones adicionales, como mover los dos grados de libertad del cuello con una velocidad determinada, o mover los dos grados de libertad en la misma función a posiciones distintas.

Los párpado se gobiernan mediante la clase *CactuadorCabeza*, que originalmente era la que controlaba también el cuello. La clase construye un objeto que puede mover cualquiera de los párpados, o los dos a la vez. Las funciones de esta clase que usaremos son:

- Función para inicializar los párpados, y permitir que puedan moverse.
- Función para mover el párpado izquierdo a una posición.
- Función para mover el párpado derecho a una posición
- Función para leer la posición actual de ambos párpados.

Esta clase C++ contiene algunas funciones adicionales, como mover los dos grados de libertad del cuello con una velocidad determinada, o mover los dos grados de libertad en la misma función a posiciones distintas.

5.3.- Interpolación mediante splines.

5.3.1.- Definición de curvas spline.

Una spline es una curva diferenciable definida en proporciones mediante polinomios. Es muy común utilizar las splines en interpolación, porque da buenos resultados utilizando solamente polinomios de bajo grado, por lo que se evitan las oscilaciones. Otras de las grandes ventajas de este método de interpolación son la estabilidad y la facilidad de cálculo. [13] [14]

La interpolación mediante splines se basa en dividir el intervalo a interpolar en pequeños subintervalos. Cada uno de esos subintervalos, se interpola usando polinomios de tercer grado como máximo. Los coeficientes de ese polinomio se eligen para que satisfagan ciertas condiciones. Las condiciones generales son que la función resultante (la curva spline propiamente dicha) sea continua y pase por los puntos

requeridos en la interpolación. Otras condiciones pueden ser continuidad en las derivadas o linearidad en la función entre los distintos nodos.

Existen numerosos tipos de interpolación spline. En el presente trabajo se ha decidido utilizar la interpolación linear. La interpolación linear se caracteriza porque los polinomios que unen los subintervalos de la interpolación son del tipo P(x) = ax + b.

La interpolación spline linear es el caso más sencillo de interpolación spline. Es sencilla de cálculo, y tiene la gran ventaja de que mantiene la monotonía de los puntos, por lo que se evitan movimientos no deseados en el robot. Como desventaja, es una interpolación menos precisa, y por lo general no es derivable en algunos puntos.

5.3.2.- Biblioteca Alglib.

Alglib es una biblioteca para realizar operaciones de análisis numérico y procesamiento de datos. Es compatible, entre otros lenguajes, con C++ y funciona, entre otros sistemas operativos, en Linux. Es portable, sencilla de usar, de código abierto, y contiene la operación de interpolación spline, que es para lo que es necesario usarla en el presente trabajo.

Existen otras alternativas a Alglib, siendo las más populares: NAG Numerical Library, que se descarta por ser de pago. Armadillo, que no contiene un módulo específico de interpolación spline, o Blitz++ que se descarta por estar peor documentada, y necesitar un parche adicional para la interpolación spline. En general, se escoge la biblioteca Alglib porque demuestra ser la de implementación más sencilla, y ofrece resultados satisfactorios.

Para usar la biblioteca Alglib necesitamos compilar ciertos paquetes, que vienen en forma de archivos .cpp y .h, y añadirlos al programa. No es necesario compilar todos los paquetes de la biblioteca. Los paquetes que hay que compilar para realizar interpolación spline son:

- alglibmisc.cpp
- integration.cpp
- interpolation.cpp

- linalg.cpp
- optimization.cpp
- solvers.cpp
- specialfunctions.cpp
- alglibinternal.cpp
- ap.cpp
- data.h
- stdafx.h

Para compilarlos, los añadimos en un fichero CMakeLists.txt con la instrucción add_maggie_lib. La biblioteca Alglib carece de ficheros Makefile o CMake propios.

La biblioteca Alglib incluye tipos de dato y funciones propios. En la descripción del programa y sus funciones se hace una descripción breve de los tipos de dato y funciones de la biblioteca Alglib que se utilzan.

5.4.- Uso de vectores.

Para trabajar y operar con los puntos, utilizaremos vectores que almacenen esa información. En C++, los vectores son contenedores dinámicos pertenecientes a la biblioteca STL. Pueden almacenar grandes cantidades de cualquier tipo de dato definido. Son una clase con funciones definidas que pueden recordar a un array. Sin embargo los vectores son más seguros y versátiles cuando se usan sus funciones miembro; por ejemplo, tratar de acceder a una posición más allá del final del vector, lanzará una excepción, en vez de terminar el programa como haría un array. Para usar vectores, debemos incluir en el programa el header #include <vector>. [15]

Para la ejecución de este programa, existe la posibilidad de usar arrays en vez de vectores, o incluso de utilizar durante todo el programa variables del tipo de dato *real_1d_array*, pertenecientes a la biblioteca Alglib, y similares a los arrays comunes, y que será necesario crear para el cálculo de la interpolación spline.

Se descarta el uso de arrays porque los vectores tienen mayor versatilidad. Nos permite operar con ellos sin conocer el tamaño final del vector y añadir elementos al mismo tiempo que aumentamos su tamaño mediante la función *push_back()*.

Se descarta el uso de *real_1d_array* porque, del mismo modo que los arrays normales, son menos versátiles que los vectores. Además, *real_1d_array* es un tipo de dato demasiado específico, se necesita el uso de la biblioteca Alglib, y puede resultar muy problemático a la hora de integrarlo con otras funciones del robot.

Después del cálculo de la spline, el vector de tiempo de cada articulación deja de ser necesario; de hecho, no se usa en la función encargada de mover las articulaciones y ejecutar el gesto del robot. Sin embargo se decide conservar, ya que sirve para mantener constancia de en qué momento se realiza cada movimiento, lo cual puede resultar útil en futuras aplicaciones.

También se estudió la posibilidad de utilizar un vector de vectores, con un elemento que almacene las posiciones y otro que almacene los tiempos. Incluso un único vector que almacene los vectores de todas las articulaciones. Sin embargo, se descarta porque esto restaría cierta versatilidad y flexibilidad al programa sin conseguir ningún beneficio extra.

5.5- CMake.

CMake, abreviatura de “cross platform make” es una herramienta multiplataforma para la generación o automatización de código. Es una suite de más alto nivel que el sistema make común de Unix, que es parte del compilador GCC mencionado en el apartado anterior. [11] [16]

CMake genera makefiles nativos, llamados CMakeLists.txt que pueden usarse en el entorno de desarrollo deseado. El proceso se controla creando uno o más ficheros CMakeLists.txt en cada directorio y subdirectorio. Cada fichero CMakeLists.txt se compone de uno o más comandos que describen algún aspecto de la compilación. CMake permite el uso de comandos definidos por el usuario.

Para compilar archivos en AD, debemos, en primer lugar, tener un archivo CMakeLists.txt en el directorio que deseamos compilar. Este archivo podemos crearlo o podemos generarlo al ejecutar la orden *ccmake* (ver más adelante). En este proyecto se compilarán dos tipos de archivos: bibliotecas, que serán la clase para realizar un gesto a partir de un fichero de gesto, la clase para realizar un gesto a partir de un archivo de forma, y la biblioteca Alglib; y tests, que serán los archivos de prueba para comprobar el funcionamiento de la clase.

Para compilar una biblioteca, se utiliza el siguiente código:

```
add_maggie_lib (nombre que tendrá la biblioteca
SOURCES archivos fuente .cpp
LINK bibliotecas de las que depende. Opcional.)
```

Para compilar un test, se utiliza el siguiente código:

```
add_maggie_test (nombre que tendrá la biblioteca
SOURCES archivos fuente .cpp
LINK bibliotecas de las que depende. Opcional.)
```

Los archivos fuente, *sources*, deben estar en el mismo directorio que el fichero CMakeLists.txt que los referencia.

Adicionalmente, para que el compilador lea el fichero CMakeLists.txt y lo compile convenientemente, debe incluirse el directorio donde está ubicado dicho fichero en el fichero CMakeLists.txt de la carpeta *manager/cmake.* Si hay una carpeta incluida en este directorio sin fichero CMakeLists.txt, se creará uno al ejecutar la orden *ccmake* (ver más adelante).

Para compilar debemos crear una carpeta BUILD, o utilizar una que ya esté creada. Desde esa misma carpeta, utilizamos el comando *ccmake /path/to/manager/cmake* donde */path/to/manager/cmake* es la ubicación del directorio cmake dentro de manager.

Se nos abrirá una lista de opciones que podemos compilar o no con las etiquetas ON y OFF. Se pueden cambiar de ON a OFF y viceversa al pulsar enter sobre ellas. Al compilar, solo se compilarán aquellas opciones con la etiqueta ON.

Una vez se hayan elegido las opciones que se van a compilar, presionar “c” realizará la configuración, y aparecerán los errores si los hay. Presionar “e” para salir. A continuación, presionar “g” generará los Makefiles y presionando “e” una vez más, se saldrá.

Después, cuando se ejecute el comando *make* en el directorio BUILD, se compilarán todos los archivos que se hayan modificado desde la última vez que se ejecutó *make* (todos si es la primera vez que se hace). Todas las aplicaciones construidas se almacenan en *BUILD/bin* y todas las bibliotecas en *BUILD/lib.*

6.- Desarrollo e implementación.

6.1.- Ficheros de gesto y ficheros de forma.

Una parte importante del programa se basa en la extracción de los puntos del gesto o de la forma abstracta de un fichero. A continuación se detalla la estructura de estos ficheros.

6.1.1.- Ficheros de gesto.

Los gestos que pueden ser ejecutados se almacenan en un fichero de gesto. Un fichero de gesto contiene la información indispensable para llevar a cabo el gesto deseado. La extensión de los ficheros será habitualmente .xml, aunque para la edición y pruebas de los gestos se podrá usar un fichero de texto con extensión .txt. La estructura de los gestos deberá ser siempre la misma, como se muestra en la siguiente figura:

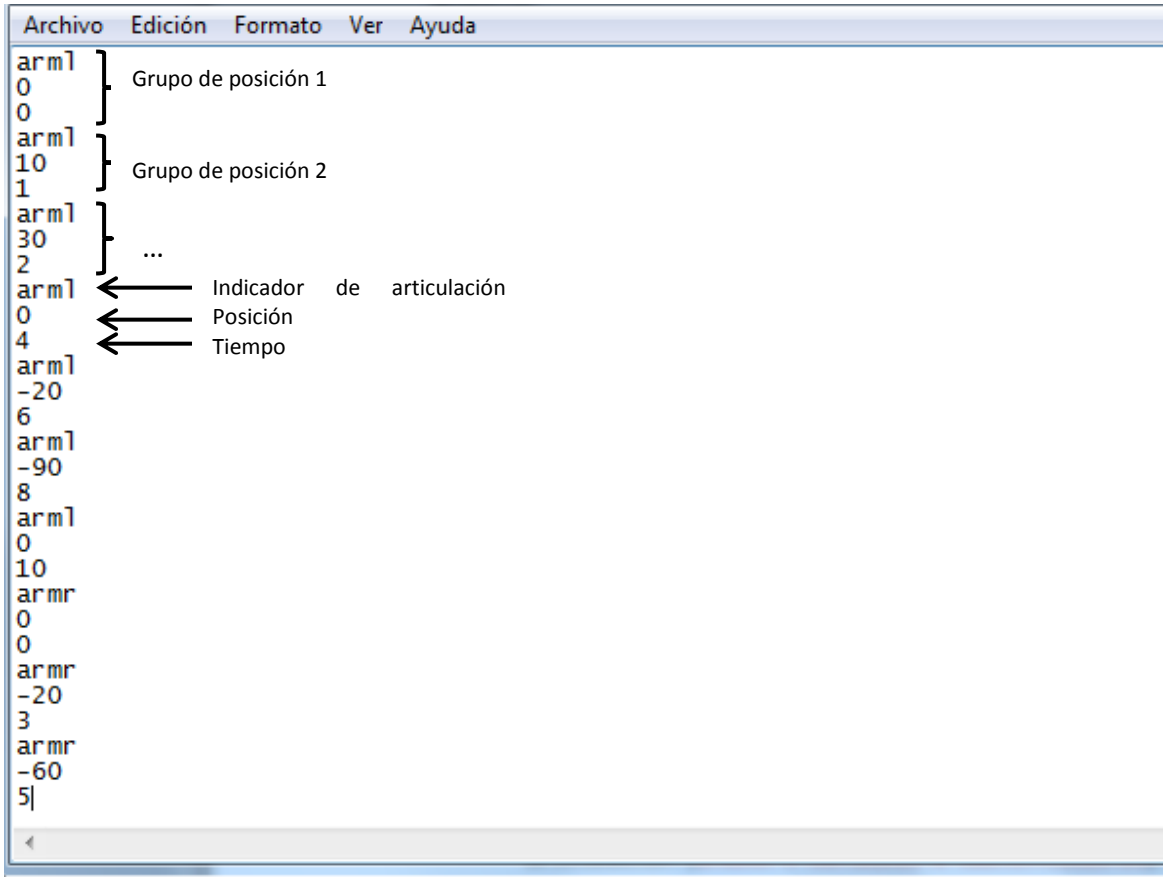


Figura 18: Estructura de un fichero de gesto.

Los ficheros de gesto se forman a partir de grupos de posición. Cada grupo se forma con tres líneas independientes de texto en el siguiente orden:

- La primera línea está compuesta de caracteres alfabéticos, y debe corresponderse a alguna de las cadenas correspondientes a un indicador de articulación (Consultar Anexo I: Identificadores de articulación). Esta línea es la que indica a qué articulación se refieren la posición y tiempo del grupo. En caso de que la cadena de caracteres de la línea no coincida con alguno de los indicadores de articulación, el sistema ignorará todo el grupo.

- La segunda línea está compuesta por caracteres numéricos. Indica la posición a la que se moverá la articulación indicada en la línea anterior. Los números pueden ser enteros o decimales y positivos o negativos.

- La tercera línea está compuesta también por caracteres numéricos. Indica el momento, expresado en segundos, en el que la articulación de la primera línea se moverá a la posición de la segunda línea. Los números pueden ser enteros o decimales y deben ser positivos o cero.

Es imperativo que la información en el grupo siga este orden para que el gesto se desarrolle según lo deseado. Si no se desea que una articulación realice movimiento alguno, basta con no incluirla en el fichero de gesto; toda articulación que no sea incluida en el fichero de gesto permanecerá estática durante la ejecución del gesto.

Si se decide incluir una articulación en el archivo de gesto, deben seguirse dos reglas adicionales para que no se produzca un error con la biblioteca Alglib al calcular la interpolación spline (Se detalla este error en el apartado 6.2.1.2). La primera de ellas, es que deben incluirse al menos dos puntos de posición por articulación. Esto es, en realidad, algo obvio, puesto que es imposible generar un movimiento a partir de un único punto. La segunda de ellas, es que, en la misma articulación, no deben incluirse dos posiciones en dos grupos distintos para un mismo momento de tiempo. También resulta evidente, puesto que una articulación no puede situarse en dos puntos distintos en el mismo instante.

No es necesario que los grupos de posición estén agrupados por articulaciones u ordenados temporalmente; el programa funcionará perfectamente clasificándolos por articulaciones en la función que toma los valores de un fichero y los guarda en un vector y los puntos quedarán ordenados tras la función que realiza el cálculo de la interpolación spline. Sin embargo, sí que es recomendable mantener un orden para facilitar la lectura, comprensión y edición de ficheros de gestos.

6.1.2.- Ficheros de forma

Las formas abstractas se almacenan en un fichero de forma. Un fichero de forma es una combinación de posiciones y tiempos que podrán convertirse en un gesto tras procesarlo. La extensión de dichos ficheros será habitualmente .xml, aunque para la edición y pruebas de las formas abstractas se podrá usar un fichero de texto con extensión .txt. La estructura de los ficheros de forma debe ser siempre la misma, tal y como se muestra en la siguiente figura.

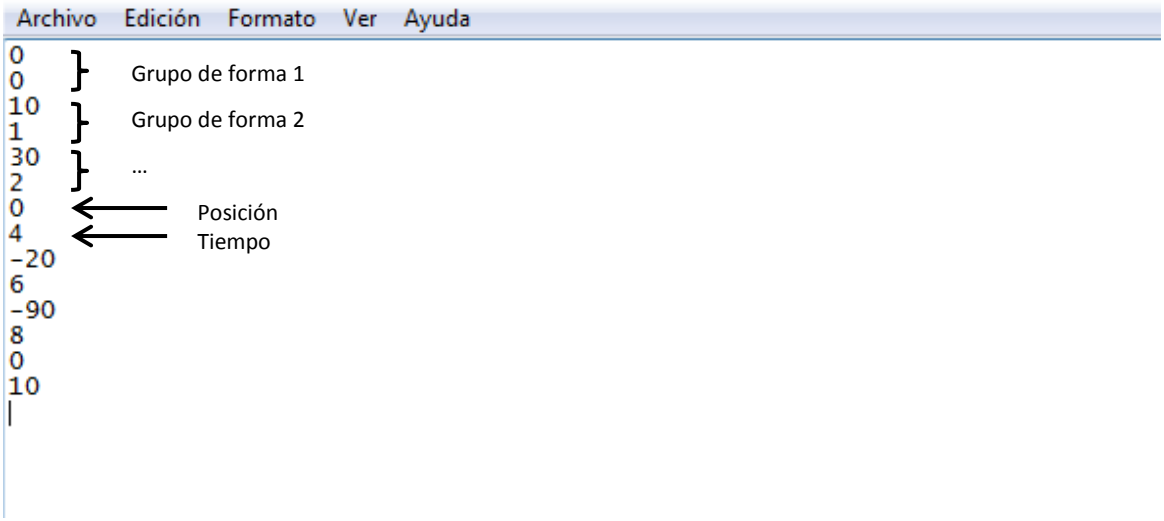


Figura 19: Estructura de un fichero de forma.

Las formas abstractas son similares a los gestos, con la diferencia de que las formas abstractas son universales para todos los grados de libertad. Debido a esta universalidad, no necesitan tener identificador de articulación; posteriormente, al procesarlas, se les asignará a la articulación que va a ejecutarlas. Análogamente a los ficheros de gesto, los datos de un fichero de forma se agrupan en “grupos de forma”.

Los grupos tendrán solamente dos líneas de texto compuestas por caracteres numéricos.

- La primera línea indica la posición a la que se moverá el grado de libertad. Los números pueden ser enteros o decimales y positivos o negativos

- La segunda línea indica el momento temporal en el que el grado de libertad se moverá a la posición indicada anteriormente. Los números pueden ser enteros o decimales y deben ser positivos o cero.

Como los ficheros de gesto, es imperativo que la información en el grupo de forma siga ese orden para que el futuro gesto se desarrolle según lo deseado.

También deben seguirse las dos reglas de los ficheros de gesto para que no se produzca un error con la biblioteca Alglib al calcular la spline (Se detalla este error en el apartado 6.2.1.2). La primera de ellas, es que deben incluirse al menos dos grupos de forma. Esto es, en realidad, algo obvio, puesto que es imposible generar un movimiento a partir de un único punto. La segunda de ellas, es que no deben incluirse dos posiciones en dos grupos distintos para un mismo momento de tiempo. También resulta evidente, puesto que una articulación no puede situarse en dos puntos distintos al mismo tiempo.

Debe tenerse en cuenta que la posición de los grupos de forma se expresa de manera relativa y la forma abstracta es la de amplitud máxima posible. Es decir, la posición con mayor valor absoluto corresponderá, cuando se convierta a un gesto, a la amplitud máxima del grado de libertad (si es positiva) o a la amplitud mínima (si es negativa). Debe tenerse en cuenta que la amplitud mínima no tiene por qué ser igual a la posición cero o de reposo para grados de libertad con posiciones negativas. El resto de posiciones serán relativas a esta posición máxima. Por ejemplo, si tenemos las posiciones *10, -20, 30,* el valor 30 será la posición máxima de la articulación, todas las posiciones serán relativas a este máximo y estarán entre -30 y 30. La posición *10* estará a 4/6 sobre el mínimo (o 1/3 sobre el punto medio) y la posición -20 estará a 1/6 sobre el mínimo (o 2/3 bajo el punto medio).

Algo análogo ocurre con los tiempos. El mayor valor de tiempo en un fichero será el tiempo máximo del gesto, y el resto de tiempos serán relativos a este.

Debe tenerse en cuenta que posteriormente puede aplicarse un bias para normalizar tanto la amplitud como longitud temporal. Para más información sobre cómo se normalizan las formas, consultar el apartado 6.2.2.2.

Como en los ficheros de gesto, los grupos de los ficheros de forma no tienen por qué mantener un orden determinado, aunque resulta recomendable para su lectura, interpretación y edición. También es recomendable que la escala que mantengan sea sencilla, por ejemplo, haciendo que el máximo valor absoluto sea 1 o 100.

6.2.- Descripción del cuerpo del programa y sus funciones.

El objetivo principal del presente proyecto es, como hemos adelantado, transformar un fichero con información de unos puntos en movimiento de las articulaciones del robot Maggie. Existen dos maneras de hacerlo, la primera de ellas es a través de un fichero de gesto, que contiene la información sobre cómo van a moverse cada una de las articulaciones del robot durante un tiempo determinado. La segunda es a través de un fichero de forma, que contiene la información de una forma que podrá ser ejecutada por cualquiera de las articulaciones de Maggie, una vez se haya procesado.

Para cada una de esas dos vías de desarrollo, existe una clase en C++. Una clase en C++ es un tipo de dato, similar a una estructura, pues consiste en una agrupación de diferentes tipos de datos, llamados miembros de la estructura. La clase se diferencia de la estructura en que ésta, además, puede tener funciones como miembros de la clase.

La clase para desarrollar un gesto a partir de un fichero de gesto se llama *splinegesture*. La clase para desarrollar un gesto a partir de un fichero de forma se llama *splineform*. Ambas clases comparten dos funciones que son prácticamente idénticas, y una que es relativamente similar. Además la clase *splineform* tiene dos funciones adicionales.

Debido a la similitud en la construcción de ambas clases, se plantea la posibilidad de combinarlas en una única clase. Sin embargo, ambas clases se van a usar para fines distintos; algunas funciones de la propia clase *splineform* pueden usarse en robots

distintos, cosa que no tendría mucho sentido con la clase *splinegesture*. A cambio lo único que se gana es reducir ligeramente el código. Por tanto se decide mantener ambas clases.

A continuación describimos las dos vías de trabajo en las que se basa este proyecto, usando como guía las dos clases mencionadas anteriormente y sus funciones.

6.2.1.- Ejecución de un gesto a partir de un fichero de gesto. La clase splinegesture.

La clase que puede ejecutar gestos a partir de un fichero de gesto contiene tres funciones que permiten:

- Leer un gesto de un archivo y guardarlo en sus respectivos vectores, mediante la función *filetovector().*

- Hacer una interpolación spline sobre los puntos de ese gesto, mediante la función *splinecalc().*

- Convertir los puntos de la interpolación en movimiento de Maggie, mediante la función *gesturemovement().*

El orden lógico de actuación es primero leer los puntos del archivo y guardarlos en vectores, después realizar la interpolación spline y por último, el movimiento del robot, aunque es posible hacer variaciones. Por ejemplo, podría saltarse la parte de extraer los puntos de un fichero si se desea recoger los puntos de los vectores de un lugar que no sea un fichero de gesto. También podría saltarse la función que realiza la interpolación spline si no se desea hacer la interpolación de los puntos (recordemos que el gesto una caracterísitica de los gestos era que no era necesario procesarlos, por lo que desde el primer momento pueden ejecutarse), o se desea realizar otra operación sobre ellos. Además, existe la opción de variar un gesto durante la ejecución del mismo. Esto haría que, durante la función de ejecución del gesto, y antes de terminar esta ejecución, se volviese a llamar a las funciones anteriores, y después continuar con la ejecución de la propia función de ejecución del gesto. Este proceso

puede realizarse tantas veces como se desee dentro de la función de ejecución del gesto, y se detalla en el capítulo 6.2.1.3.

En el siguiente diagrama muestra el funcionamiento normal de la clase, el orden lógico de actuación. Los rectángulos son las funciones, y las flechas gruesas el flujo de la ejecución. Los círculos representan información y las flechas punteadas el flujo de la información.

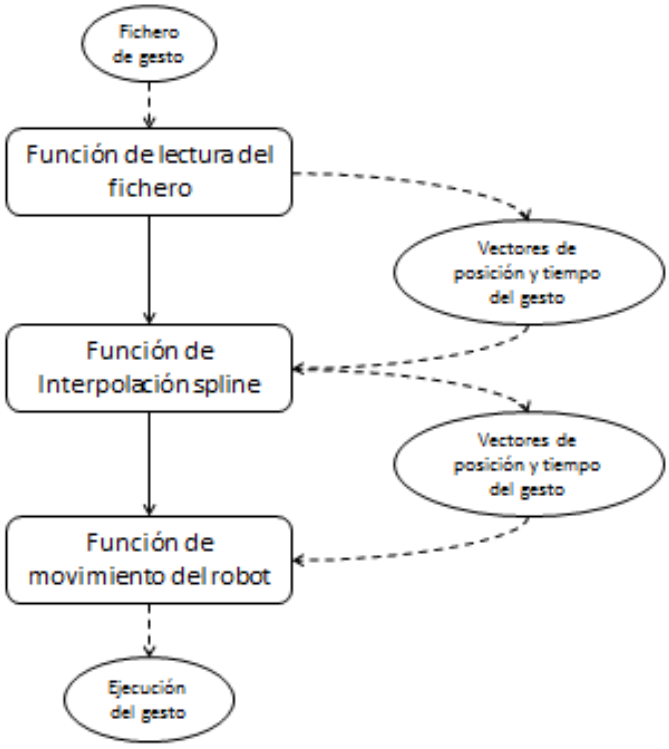


Figura 20: Diagrama de funcionamiento de la clase splinegesture.

La clase *splinegesture* se encuentra descrita en el archivo fuente *splinegesture.cpp*, con declaración previa en *splinegesture.h*. Para compilar esta clase se debe incluir en el archivo CMakeLists.txt, ubicado en la misma carpeta, usando la instrucción *add_maggie_lib.*

Para trabajar y operar con los puntos se utilizarán vectores que almacenen esa información. Existen dos vectores para cada grado de libertad, uno de ellos almacena la posición y otro el tiempo. Los pares de puntos posición-tiempo tienen la misma posición de elemento en sus respectivos vectores. Así, por ejemplo, el elemento 3 del

vector de posición del brazo izquierdo es el punto al que se tiene que mover el brazo izquierdo en el momento que indica el elemento 3 del vector de tiempo del brazo izquierdo.

En este programa todos los vectores son de tipo *double*, es decir, los datos que almacenarán serán de tipo *double*. El nombre de las variables vector de posición es: *pose + (indicador de articulación)*. El nombre de las variables vector de tiempo es: *time + (indicador de articulación)*. Por ejemplo, para el brazo izquierdo, el vector de posición es *posearml* y el de tiempo, *timearml*. Para una lista completa de los indicadores de articulación, consultar Anexo I: Identificadores de articulación.

6.2.1.1.- Lectura del fichero. Función filetovector()

Función que lee los datos de un fichero de gesto y los graba en los vectores de posición y tiempo de la articulación correspondiente.

La función comienza creando una variable *ifstream* para trabajar con el fichero y abriendo con ella un archivo de gesto. En caso de que el archivo no pueda abrirse, se informará por pantalla al usuario, se finalizará la ejecución de la función y se pondrá la variable *faliure* con valor 1. La variable *faliure* avisa de la existencia de un error. Mientras sea 0, significa que todo funciona correctamente. Un valor de 1 indica que ha habido un fallo en la lectura del fichero.

Una vez el archivo esté abierto, se irán leyendo las filas del fichero de tres en tres hasta llegar al final de archivo (*eof, end of file*). Las filas se van leyendo de tres en tres para leer de una vez todo un "grupo de posición". La primera fila del grupo es el indicador de articulación y señala la articulación que deberá ejecutar el movimiento, la segunda fila es la posición a la que deberá moverse y la tercera el momento temporal en el que deberá hacerlo.

La función *filetovector()* identifica a través de un *strncmpr()*, o función que compara dos cadenas de caracteres, entre el indicador de articulación del grupo y todos los posibles indicadores de articulación, a qué articulación se refiere. Si el indicador del grupo coincide con algún identificador de articulación, convertirá las dos líneas

siguientes, a tipo de dato *double* y grabará en los vectores correspondientes de posición y tiempo la posición y tiempo que estaban escritas en el grupo del fichero. Si no coincide con ninguno, no se grabará nada.

Por ejemplo, si el grupo que hemos leído contiene la siguiente información:

arml
10
2

La función *filetovector()* identificará que el grupo se refiere a la articulación *arml* (brazo izquierdo) por tanto, convertirá el dato *10* a tipo *double* y lo grabará en la siguiente posición del vector *posearml*. Después convertirá el dato *2* a tipo *double* y lo grabará en la siguiente posición del vector *timearml*.

La información se graba en los vectores mediante la función de la clase vector *push_back()*. Esta función amplía en uno el tamaño del vector y después graba el dato introducido en la última posición, que será la recién creada. Inicialmente el tamaño de los vectores es cero, y su tamaño final será igual al número de datos que se le hayan introducido.

Si se graba información en los vectores de una articulación determinada, se activará la variable miembro de la clase *bool_(articulación).* Esta variable es 0 (‘false’) si la articulación no va a moverse, a participar, durante el gesto, y 1 (‘true’) si la articulación participa en el gesto. En el momento en el que se graba información en el vector de la articulación, se considera que esta va a formar parte del gesto, y por tanto, su variable *bool_(articulación)* tomará el valor 1. Siguiendo con el ejemplo anterior, la variable *bool_arml* tomaría automáticamente el valor 1.

El proceso de leer un grupo de posición, identificar a qué articulación se refiere, convertir a *double* y grabar en los vectores, se repite cíclicamente hasta que se llega al final del archivo; incluido cuando el indicador del grupo no coincide con ningún indicador de articulación. Solo en el caso de que se haya llegado al final del archivo, se cerrará el fichero y terminará la función con los datos almacenados en los vectores.

Además, durante la lectura del fichero, la función comprueba, para cada tiempo leído, si es el tiempo máximo almacenado en el vector de tiempo de la articulación Si lo es, lo almacena en la variable *gesture_time_(articulación)*. Existe una variable *gesture_time_(articulación)* para cada articulación. Este paso será importante para el cálculo de la spline.

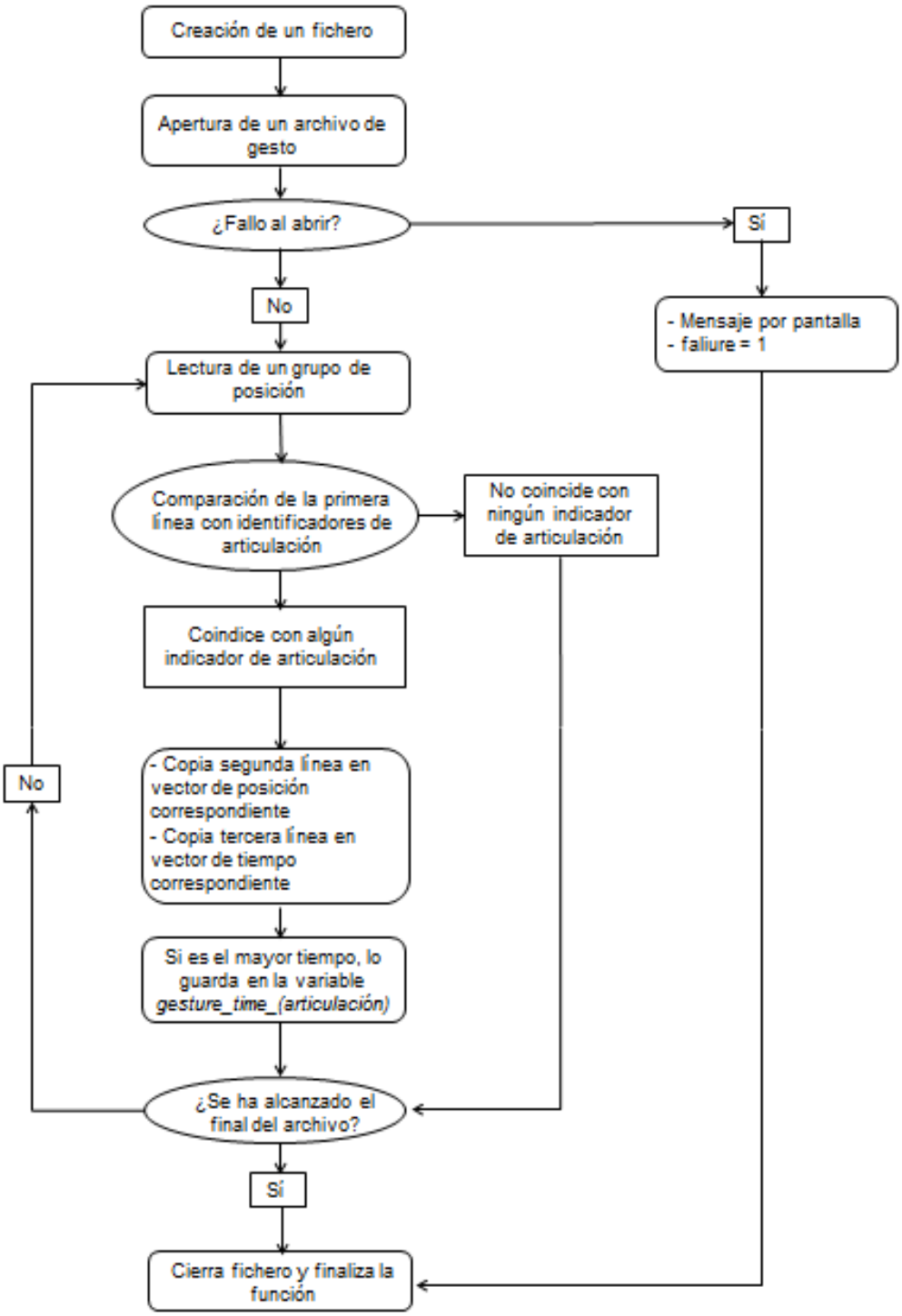


Figura 21: Diagrama de flujo de la función filetovector.

6.2.1.2.- Cálculo de la spline. Función splinecalc()

Funcionamiento de la función splinecalc()

La función *splinecalc()* toma los valores de los vectores de posición y tiempo de cada articulación y hace con ellos una interpolación spline. Después guarda los puntos de la spline en estos mismos vectores con un intervalo de 0.03 segundos entre ellos.

Las curvas spline se generan usando las funciones de la biblioteca Alglib, descrita en el capítulo 5.3.2. Para calcular splines, la biblioteca Alglib necesita que los datos le sean presentados en variables del tipo *real_1d_array* incluidas en la propia biblioteca Alglib. Estas variables funcionan de forma similar a un array normal. Del mismo modo que existe un vector para posición y uno para tiempo por cada articulación, habrá una variable del tipo *real_1d_array* para posición y una para tiempo por cada articulación. Estas variables se nombran igual que los vectores, añadiendo la palabra array al final, es decir, para posición: *pose(articulación)array,* y para tiempo: *time(posición)array.* Por ejemplo, para el brazo izquierdo, son *posearmlarray* y *timearmlarray.*

De forma adicional, la biblioteca Alglib necesita una variable del tipo *spline1dinterpolant* donde almacenar la spline. Esta variable también forma parte de la biblioteca Alglib. Los nombres de estas variables serán *spline_(articulación).* Por ejemplo, para el brazo izquierdo es *spline_arml.*

Lo primero que hará la función para calcular la interpolación spline será crear una variable *real_1d_array* por cada vector, es decir, dos por cada articulación, y una del tipo *spline1dinterpolant* para cada articulación.

A continuación comprobará que la articulación participa en el gesto. Esta comprobación se realiza a través de la variable *bool_(articulación)* que será 1 si la articulación participa y 0 cuando no lo hace.

En caso de que la articulación participe en el gesto, se procederá a pasar los datos desde los vectores a las variables array de la biblioteca Alglib. Estas variables están vacías y son de tamaño cero por defecto, por lo que habrá que comenzar por redimensionarlas a un tamaño idéntico al de los vectores. A continuación se copiarán

los datos de los vectores a las variables array de la biblioteca Alglib correspondientes a su tipo de vector y a su articulación.

A continuación se calcula la interpolación spline. Para ello se usa la función de Alglib *spline1dbuildlinear()* que almacenará la curva spline en la variable *spline1dinterpolant.* Para calcularla se utilizarán como argumentos, dos variables array de la biblioteca Alglib*,* además de la variable de esta misma biblioteca para almacenar curvas spline, y que será donde se guardará el resultado de la interpolación.

Queda almacenar los puntos de la curva spline de nuevo en los vectores. Para ello, el primer paso es borrar los datos actuales y redimensionar los vectores a tamaño cero. A continuación se irá grabando en cada elemento del vector de posición, un punto extraído de la variable que almacena la curva spline, cada 30 milisegundos, desde el segundo cero hasta el final del gesto. En el vector de tiempo se almacenará el tiempo, es decir, irá contando cada 30 milisegundos desde cero hasta el tiempo final del gesto.

A la hora de efectuar el movimiento de las articulaciones en el robot, se necesitará un tiempo de refresco para grabar los datos por el puerto serie. Teóricamente, para Maggie ese tiempo es de 70 milisegundos, y es el que inicialmente se usó como intervalo. Sin embargo, durante las pruebas se comprobó que era un tiempo demasiado largo y provocaba que el movimiento no fuese continuo, por lo que, se decidió bajar ese tiempo, y tras nuevas pruebas, se comprobó que funcionaba bien a 30 milisegundos con un movimiento continuo y mucho más suave.

La información se graba en los vectores mediante la función de la clase vector *push_back()*. Esta función amplía en uno el tamaño del vector y después graba el dato introducido en la última posición, que será la recién creada. Inicialmente el tamaño de los vectores es cero, y su tamaño final será igual al número de datos que se le hayan introducido, es decir, igual a los segundos que dure el gesto dividido entre 30 milisegundos.

Finalmente se iguala la variable *length* al tamaño del vector de mayor tamaño. La variable *length* almacena el tamaño del gesto, que es igual al tamaño del vector de mayor tamaño, y será importante a la hora de definir cuándo termina el movimiento del gesto durante su ejecución en la función que realiza el movimiento del robot.

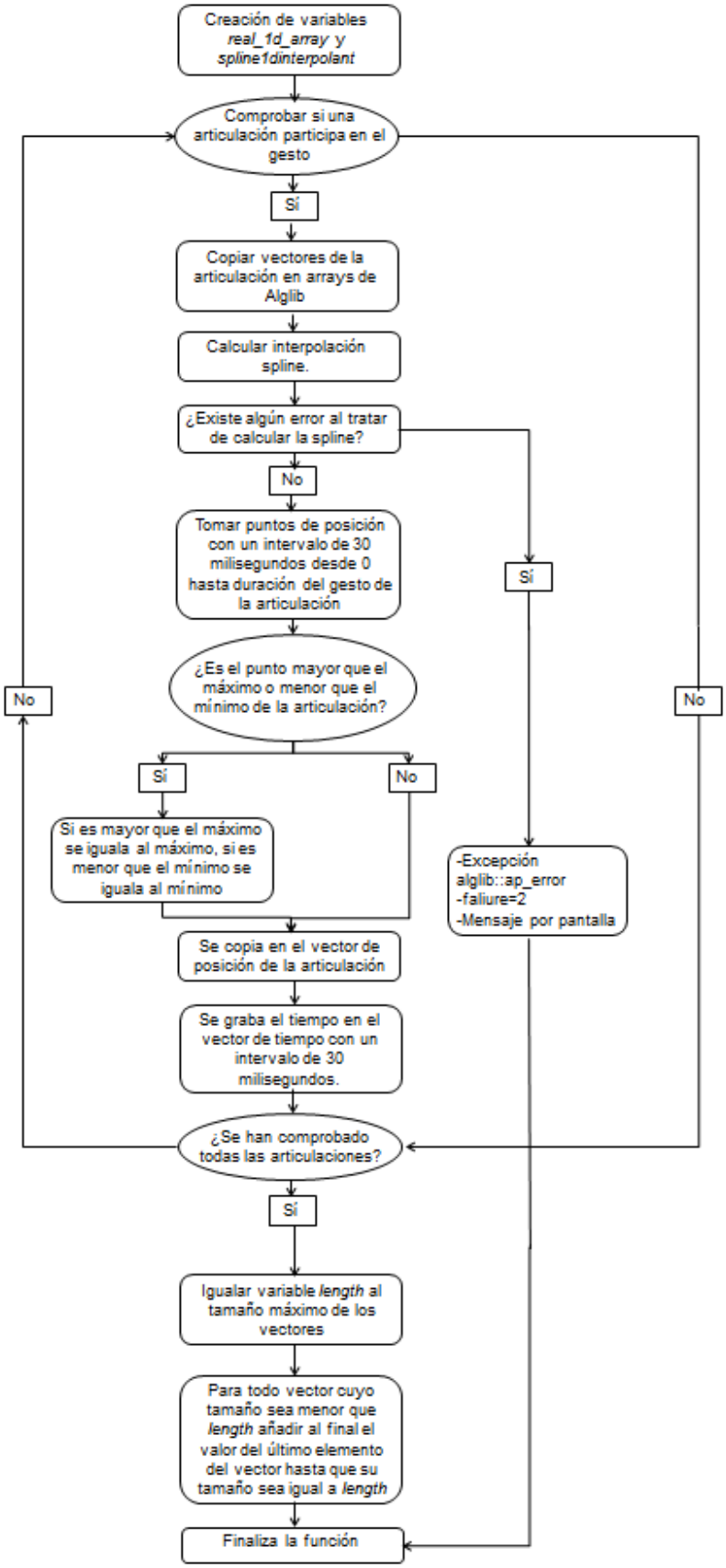


Figura 22: Diagrama de flujo de la función splinecalc().

La interpolación spline está obligada a que la curva pase necesariamente por los puntos entregados para interpolar, pero el comportamiento en todos los demás puntos no está sujeto a ninguna regla. Durante las pruebas se comprobó que esto podía provocar dos fenómenos. Por un lado, si se pretendía que una articulación dejase de moverse antes que alguna de las otras durante un gesto, la función seguiría guardando puntos nuevos de la curva spline y la articulación seguiría moviéndose hasta el final del gesto. Por otro lado, era posible que la interpolación devolviese puntos por encima del máximo permitido por una articulación o por debajo de su mínimo.

Solución al problema de movimiento no deseado al final del gesto.

Supongamos una articulación A que en el fichero de gestos el último punto descrito se corresponde al instante t_n, y existe otra articulación B que en el fichero de gestos el último punto descrito está en el instante t_m, siendo n<m. Durante el tiempo que transcurre desde el instante n al instante m, la interpolación de la articulación A seguirá arrojando puntos, es decir, que la articulación se seguirá moviendo durante m — n segundos aun cuando no se han definido más puntos para ella.

Para solucionar esto, y que cada articulación se detenga en el último punto definido, una posibilidad era obligar, por definición, a que en el archivo de gesto todas las articulaciones finalizasen su movimiento en el mismo instante temporal. La otra posibilidad consistía en forzar, mediante el programa, que todos los puntos que sean temporalmente posteriores al último punto definido en una articulación, sean iguales al último punto definido en la articulación. Para conseguir esto es necesario además incluir una nueva variable para cada articulación que almacene la duración total en segundos del movimiento de esa articulación en concreto, y sustituirlas por la antigua variable *gesture_time* que almacenaba el tiempo de gesto global, que era igual al tiempo máximo de las seis articulaciones. Finalmente se opta por elegir esta segunda opción, ya que da mayor libertad a la hora de crear y editar los ficheros de gesto. Se incluyen por tanto, seis variables, una por cada articulación, que almacenan la duración en segundos de cada articulación, con el nombre *gesture_time_(articulación).*

Tras el cálculo de la interpolación spline, se comprueba qué movimiento de las articulaciones es el que dura más tiempo, en segundos. El número de elementos de ese

vector será el nuevo valor de la variable *length*. Si hay alguna articulación en la que el tamaño de sus vectores sea inferior a la variable *length,* se copiará el contenido del último elemento del vector y se grabará en nuevos elementos que se añadirán al final mediante la función *push_back()* de la clase vector. Se añadirán tantos elementos como sean necesarios para que, tras la operación, el número de elementos del vector sea igual a la variable *length.* Por tanto, finalmente todos los vectores tendrán el mismo número de elementos, que será igual a la variable *length,* e igual al número de elementos que había en el vector de mayor tamaño antes de la operación.

Solución al problema de posible movimiento más allá de los límites de la articulación

Existen ya en las propias articulaciones del robot unos límites definidos que no se pueden superar. Sin embargo, para añadir seguridad, se decide incluir límites por software, evitando así posibles accidentes. Además, esto da la opción de poder reducir los límites en caso de que se considere necesario. En el caso de la interfaz que ejecuta gestos a partir de formas abstractas, modificar estos límites puede ser útil, ya que al convertir la forma abstracta en un gesto, esta se dimensiona según los límites máximo y mínimo de la articulación.

Para evitar que la articulación pueda moverse por encima de su máximo o por debajo de su mínimo físicos, se plantean, de nuevo, dos opciones. La primera es obviar todos los puntos que estén por encima del máximo o debajo del mínimo y excluirlos del movimiento. La segunda es forzar unos límites de tal manera que toda posición por encima del máximo físico de la articulación se iguale al máximo y toda posición inferior al mínimo físico, se iguale al mínimo.

El efecto de ambas soluciones es similar, una vez que se alcance el límite, la articulación dejaría de moverse hasta que se mande una nueva posición que no sobrepase el límite. Además, ambas son también sencillas de implementar. Finalmente se decide escoger la segunda opción por ser un poco más sencilla y porque, aunque el efecto de ambas soluciones es similar, es preferible mantener un control constante de la posición de la articulación, que conseguimos enviando constantemente puntos.

Excepción de la biblioteca Alglib.

Durante las pruebas se descubre la existencia de un error en la ejecución si se pasan ciertos datos a la librería Alglib para hacer la interpolación spline. Si para una misma curva spline se pasan dos posiciones del eje Y (que en este caso representa la posición) para una misma posición del eje X (en este caso el tiempo), o bien si se manda un único punto (x,y) para interpolar, el programa lanzará una excepción *alglib::ap_error.*

Se ha realizado un manejo de excepciones para este caso. Si el programa lanza una excepción *alglib::ap_error* esta excepción será capturada, mediante un *catch,* se informará al usuario con un mensaje de error, y se pondrá el valor de la variable *faliure* a dos. La variable *faliure* indica cuándo hay un fallo. Si el valor de la variable es cero, significa que el programa funciona perfectamente. Si su valor es igual a dos, significa que ha habido un fallo al calcular la curva spline.

6.2.1.3.- Ejecución física del gesto. Movimiento del robot. Función gesturemovement()

Funcionamiento normal de la función gesturemovement.

La función gesturemovement() toma los valores de los vectores de posición y tiempo y los transforma en movimiento del robot Maggie.

Puesto que vamos a trabajar con las articulaciones de Maggie, es necesario crear un objeto de cada clase de las interfaces de movimiento de las articulaciones para poder trabajar con sus funciones y mover el robot. Aquí se ofrece una descripción rápida de las clases con las interfaces para mover los brazos, el cuello y los párpados de Maggie.

Para mover los brazos de Maggie usamos la clase *CarmBase*, descrita en *armBase.h.* Se inicializa un objeto de esta clase bajo el nombre *arms*. Después se usa la función miembro de *CarmBase*, *init()* para inicializar los servos de los brazos de Maggie y estos puedan moverse.

Para mover el cuello de Maggie usamos la clase *NeckApi*, descrita en *neck_api.h.* Se inicializa un objeto de esa clase bajo el nombre *neck.* Después se usa la función miembro de *NeckApi*, *enable()* para inicializar los servos del cuello de Maggie y este pueda moverse.

Para mover los párpados de Maggie usamos la clase *CactuadorCabeza*, descrita en *actuadorCabeza.h.* Se inicializa un objeto de esa clase bajo el nombre *eyes*. Después se usa la función miembro de *CactuadorCabeza*, *init()* para inicializar los servos de los párpados de Maggie y estos puedan moverse.

A continuación se inicia un bucle *for*, cuya duración será de tantos ciclos como indique la variable *length.* La variable *length* almacena el número de elementos del vector de mayor tamaño, y tras la función que realiza la interpolación spline de los puntos de los vectores*,* es también el número de elementos en cualquiera de los vectores de alguna articulación que participe en el gesto. Por tanto, el bucle *for* dura tantos ciclos como elementos hay en cualquiera de los vectores de una de las articulaciones que participan en el gesto.

Durante el bucle, cada ciclo se comprueba, para cada articulación, si esta participa, o no en el gesto a través de la variable *bool_(articulación)* de esa articulación. La variable *bool_(articulación)* es un *boolean* con el valor será 0 (‘false’) si la articulación no participa en el gesto, y 1 (‘true’) si sí lo hace. Si participa, la articulación se moverá a la posición que indique el elemento i del vector de posición de esa articulación, donde i es el ciclo actual en el que se encuentra el bucle *for*.

Para el brazo izquierdo, el movimiento se hará con la función *moveLeft* de la clase *CarmBase*. La posición que se envía está expresada en grados, pero la función *moveLeft* funciona con radianes, por tanto, antes de realizar el movimiento, será necesario hacer una conversión, multiplicando la posición por el número pi y dividiendo por 180. La función *gesturemovement()* no tiene ninguna instancia que muestre por pantalla la posición a la que se mueve el brazo izquierdo, porque la propia función de mover el brazo ya incluye una instancia que lo hace por defecto.

Para el brazo derecho, el movimiento se hará con la función *moveRight* de la clase *CarmBase*. La posición que se envía está expresada en grados, pero la función *moveRight* funciona con radianes, por tanto, antes de realizar el movimiento, será necesario hacer una conversión, multiplicando la posición por el número pi y dividiendo por 180. La función *gesturemovement()* no tiene ninguna instancia que

muestre por pantalla la posición a la que se mueve el brazo izquierdo, porque la propia función que mueve el brazo ya incluye una instancia que lo hace por defecto.

Para el movimiento vertical, o theta, del cuello, el movimiento se realiza con la función *move_theta* de la clase *NeckApi*. La posición que se envía está expresada en grados y la función usa grados también, por tanto no es necesario hacer ninguna conversión. A la par que el movimiento, se imprime por pantalla la posición a la que se mueve el cuello bajo el mensaje "Moving neck theta to [posición]".

Para el movimiento horizontal, o phi, del cuello, el movimiento se realiza con la función *move_phi* de la clase *NeckApi*. La posición que se envía está expresada en grados y la función usa grados también, por tanto no es necesario hacer ninguna conversión. A la par que el movimiento, se imprime por pantalla la posición a la que se mueve el cuello bajo el mensaje "Moving neck phi to [posición]".

Para el párpado izquierdo, el movimiento se realizará mediante la función *moverOjoIzquierdo* de la clase *CactuadorCabeza*. La posición que se envía está en escala de cero (cerrado) a cien (abierto), mientras que la función usa una escala de 20 (cerrado) a 140 (abierto). Por tanto habrá que realizar una conversión según la fórmula de la figura 20. Se imprime por pantalla la posición a la que se mueve el cuello bajo el mensaje "Moving Left eyelid to: [posición sin convertir]".

Para el párpado derecho, el movimiento se realizará mediante la función *moverOjoDerecho* de la clase *CactuadorCabeza*. La posición que se envía está en escala de cero (cerrado) a cien (abierto), mientras que la función usa una escala de 205 (cerrado) a 80 (abierto). Por tanto habrá que realizar una conversión según la fórmula de la figura 20. Se imprime por pantalla la posición a la que se mueve el cuello bajo el mensaje "Moving Left eyelid to: [posición sin convertir]".

$$Posición\ articulación = \left(Posición\ vector \cdot \frac{Pos_{max} - Pos_{min}}{100}\right) + Pos_{min}$$

$$Donde: Pos_{max} = Posición\ máxima\ de\ la\ articulación\ párpado$$

$$Pos_{min} = Posición\ mínima\ de\ la\ articulación\ párpado$$

Figura 23: Conversión de la posición para los párpados.

Las posiciones máxima y mínima de cada párpado las extraemos del archivo RobotConfig.h que se encuentra en la carpeta /manager/ad_core/definitions/.

Tras realizarse el movimiento de todas las articulaciones implicadas en el gesto, se efectúa una pausa de 30 milisegundos. A continuación termina el ciclo. Si es el último ciclo del bucle for, se termina el bucle y acaba la función de ejecución del gesto. Si no, comienza un nuevo ciclo del bucle *for*.

A la hora de efectuar el movimiento de las articulaciones en el robot, se necesitará un tiempo de refresco para grabar los datos por el puerto serie. Teóricamente, para Maggie ese tiempo es de 70 milisegundos, y es el que inicialmente se usó como intervalo. Sin embargo, durante las pruebas se comprobó que era un tiempo demasiado largo y provocaba que el movimiento no fuese continuo, por lo que, se decidió bajar ese tiempo, y tras nuevas pruebas, se comprobó que funcionaba bien a 30 milisegundos.

Si aún no se ha llegado al último ciclo del bucle *for*, es decir, si el número de ciclos ejecutados es menor que el valor de la variable *length*, se comienza un nuevo ciclo de ejecución del gesto y movimiento de las articulaciones implicadas. En caso contrario, el movimiento ha terminado y finaliza la función.

Cambio de gesto durante la ejecución en la función gesturemovement().

La función *gesturemovement()* incluye la posibilidad de cambiar el gesto durante la ejecución del mismo. Para ello se incluye un flag en forma de variable boolean: *change_gesture.* Durante el ciclo *for* de la función, mientras la variable *change_gesture* sea igual a 0 (‘false’) que es además su valor por defecto, el ciclo se desarrolla normalmente como acabamos de describir en el apartado anterior. Si en algún momento su valor cambia a 1 (‘true’), se interrumpe el movimiento y carga un nuevo archivo de gesto, calcula la interpolación spline de es nuevo gesto, y lo ejecuta.

Para llevar esto a cabo, la función *gesturemovement()*, en primer lugar, guarda en el elemento 0 del vector de posición de cada articulación el valor del elemento i del vector de posición de esa articulación, donde i es el ciclo actual del bucle *for* de la función de ejecución del gesto. Después cambia el tamaño del vector a un único

elemento, quedando como ese único elemento del vector de posición el elemento 0 con el valor de la posición actual que se acaba de grabar. Esto se hace para que a la hora de hacer la interpolación spline con el nuevo gesto, se tenga en cuenta la posición actual y el cambio del gesto que se estaba ejecutando al nuevo gesto sea menos brusco.

Paralelamente, guarda en el elemento 0 del vector de tiempo el valor -0.03. Después se cambia el tamaño del vector a un único elemento, quedando como único elemento del vector de tiempo el elemento 0 con el valor -0.03. Esto se hace, acompañado con lo descrito en el párrafo anterior, para que a la hora de hacer la interpolación spline con el nuevo gesto, se tenga en cuenta la posición actual de la articulación, y el cambio de un gesto al siguiente sea lo menos brusco posible. Se toma este valor porque corresponde a -30 milisegundos, es decir, al valor inmediatamente anterior a la ejecución del nuevo gesto.

A continuación resetean las variables *bool_(articulación)* a cero. Es decir, hasta que se indique lo contrario, se considera que no va a moverse ninguna articulación, de forma análoga al inicio del programa. Si el nuevo gesto va a incluir alguna de esas articulaciones, volverá a activar su valor a uno de forma natural durante la función de extracción de los puntos de un archivo de gesto.

Después se hace una llamada a la función *filetovector()* para que lea el nuevo gesto y lo guarde en los vectores de posición y tiempo, manteniendo en el elemento 0 del vector de posición el valor de la posición actual y en el elemento 0 del vector de tiempo el valor -0.03.

A continuación se llama a la función encargada de hacer la interpolación spline para que realice una interpolación para cada articulación que participa en el nuevo gesto con los datos del nuevo gesto y de la última posición del antiguo gesto. Se calcula un nuevo valor de la variable *length* que determinará la nueva duración del bucle *for* de la función que ejecuta el gesto, que es la que todavía se encuentra ejecutándose.

Se resetea el valor de la variable i del bucle *for* a cero. A efectos prácticos, esto significa que el sistema empieza una nueva cuenta desde cero de cuántos ciclos del bucle *for* ha recorrido. Debe recordarse que el bucle *for* seguirá ejecutándose hasta que la cuenta

de ciclos ejecutados sea igual a la variable *length* es decir, al número total de elementos en los nuevos vectores.

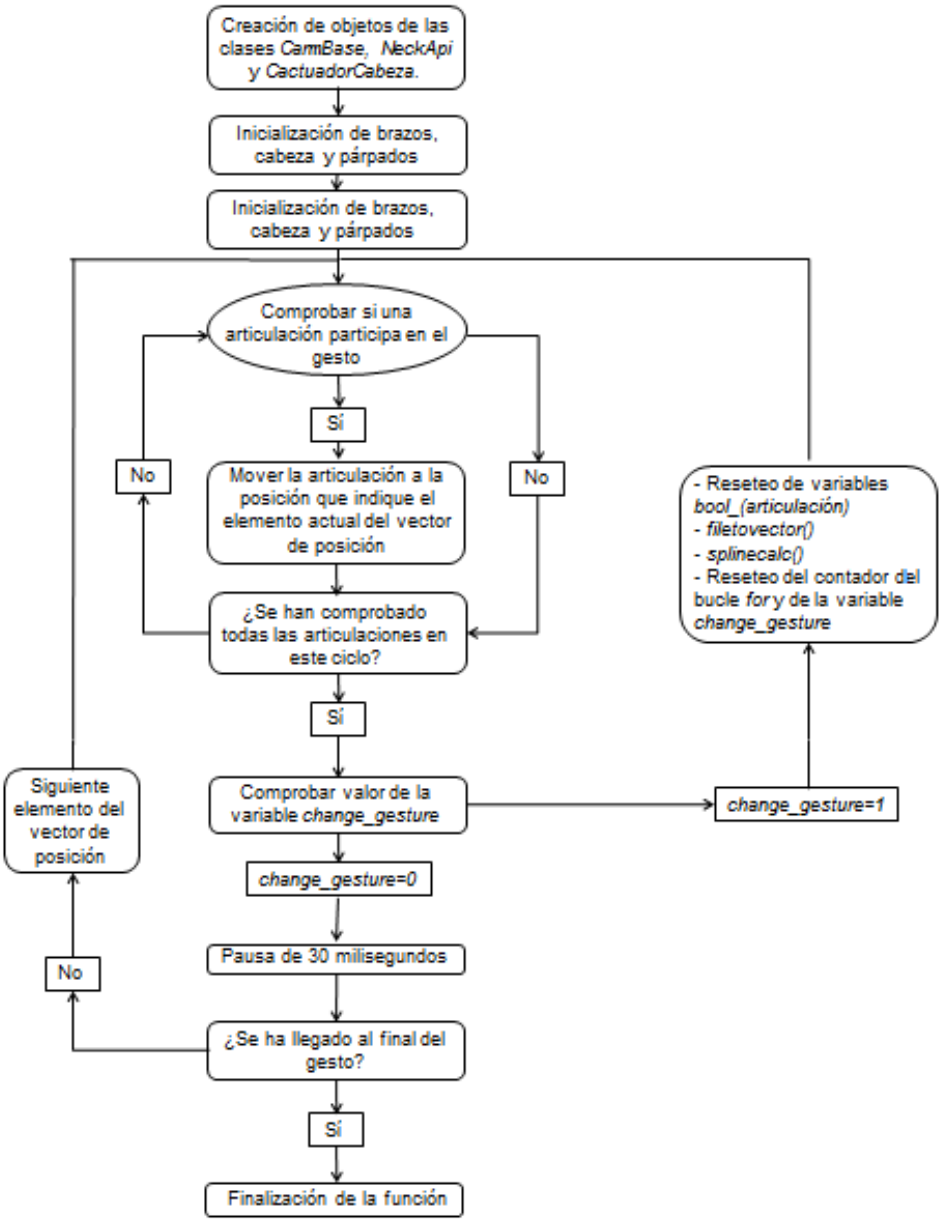


Figura 24: Diagrama de flujo de la función gesturemovement().

6.2.2. Ejecución de un gesto a partir de un fichero de forma. La clase splineform.

La clase *splineform* contiene cinco funciones que permiten:

- Leer una forma de un archivo, mediante la función *filetovector_field().*
- Normalizar la forma abstracta y modificar su amplitud y duración mediante la función *normalize_field().*
- Convertir una forma normalizada en un gesto, asociándolo a una articulación del robot mediante la función *togesture_field().*
- Hacer una interpolación spline sobre los puntos de ese gesto, mediante la función *splinecalc().*
- Convertir los puntos de la interpolación en movimiento de Maggie, mediante la función *gesturemovement().*

El orden lógico de actuación es primero lectura del fichero de forma, después normalizar la forma abstracta, convertirla en un gesto asociándola a una articulación, calcular la interpolación spline, y por último ejecutar el gesto, aunque es posible hacer variaciones. Por ejemplo, podría saltarse la función que lee el archivo de la forma abstracta si se desea recoger los puntos de los vectores de un lugar que no sea un fichero de gesto. También podría saltarse una o ambas de las funciones que normalizan la forma y calculan la interpolación spline si no se desea normalizar o aplicar un bias; o hacer la interpolación de los puntos; o se desea realizar otra operación equivalente sobre ellos. Además, existe la opción de variar un gesto durante la ejecución del mismo. Esto haría que, durante la misma función que ejecuta el gesto y antes de terminarla, se volviese a llamar a las funciones anteriores, y después continuar con la ejecución de la propia función que ejecuta el gesto. Este proceso puede realizarse tantas veces como se desee dentro de la función *gesturemovement()*, y se detalla en el capítulo 6.2.1.3.

En el siguiente diagrama muestra el funcionamiento normal de la clase, el orden lógico de actuación. Los rectángulos son las funciones, y las flechas gruesas el flujo de la ejecución. Los círculos representan información y las flechas punteadas el flujo de la información.

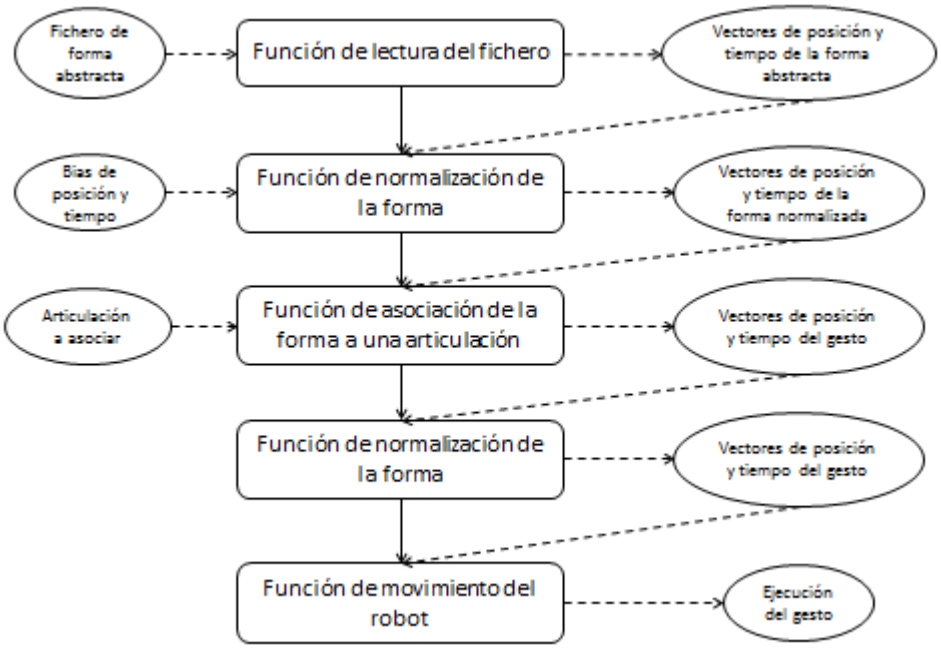


Figura 25: Diagrama de flujo de la clase splineform.

La clase *splineform* se encuentra descrita en el archivo fuente *splineform.cpp*, con declaración previa en *splineform.h*. Para compilar esta clase se debe incluir en el archivo CMakeLists.cpp, ubicado en la misma carpeta, usando la instrucción *add_maggie_lib.*

La clase *splineform* es similar a la clase *splinegesture*. Ambas comparten la función que calcula la interpolación spline que es idéntica en ambos casos, y la función que ejecuta el gesto que es idéntica en ambos casos salvo en una línea. Además la clase *splinegesture* tiene la función miembro que lee los datos a partir de un fichero de forma, que es similar a la función que lee los datos de un fichero de gesto de la clase *splinegesture.* Por último, la clase *splineform* tiene las funciones miembro de normalizar la forma abstracta, y convertir la forma en un gesto, que no se encuentran ni tienen una función análoga, en la clase *splinegesture*.

Para trabajar y operar con los puntos se utilizarán vectores que almacenen esos datos. Existen dos tipos de vectores, de posición y de tiempo. Los pares de puntos posición-tiempo tienen la misma posición del vector. Así, por ejemplo, la posición 3 del vector de posición del brazo izquierdo es el punto al que se tiene que mover el brazo izquierdo en el momento que indica la posición 3 del vector de tiempo del brazo izquierdo. Existe

un vector de posición y uno de tiempo para cada grado de libertad, además de un vector de posición y uno de tiempo para la forma sin procesar.

En este programa los vectores son de tipo *double*, es decir, los datos que almacenarán serán de tipo *double*. El nombre de las variables vector de posición es: *pose(articulación)*, a excepción del vector de posición de la forma sin procesar, que será *poseform*. El nombre de las variables vector de tiempo es: *time(articulación)*. A excepción del vector de tiempo para la forma sin procesar, que será *timeform.* Por ejemplo, para el brazo izquierdo, el vector de posición es *posearml* y el de tiempo, *timearml*.

6.2.2.1.- Lectura del fichero. Función filetovector_field()

Función que lee los elementos de un fichero de forma y los graba en un vector de posición y uno de tiempo.

La función comienza creando una variable del tipo *ifstream* llamada *raw_points* para trabajar con el fichero, y abriendo un archivo con los datos de la forma. En caso de que el archivo no pueda abrirse, se informará por pantalla al usuario, se finalizará la ejecución de la función y se pondrá la variable *faliure* con valor 1. La variable *faliure* avisa de la existencia de un error. Mientras sea 0, significa que todo funciona correctamente. Un valor de 1 indica que ha habido un fallo en la lectura del fichero.

Una vez el archivo esté abierto, se irán leyendo, a través de un ciclo *while,* las filas del fichero de dos en dos hasta llegar al final de archivo (*eof, end of file*). Las filas se van leyendo de dos en dos para leer de una vez todo un “grupo de forma”. La primera fila del grupo indica la posición a la que tendrá que moverse la futura articulación que ejecute la forma y la segunda el momento temporal en el que deberá hacerlo.

A continuación convertirá ambas filas a tipo de dato *double* mediante la función *atof()*, perteneciente a la biblioteca estándar *stdlib.h.* Y grabará en el vector de posición de la forma, *poseform* la primera línea y en el vector de tiempo de la forma, *timeform* la segunda línea.

Por ejemplo, si el grupo que hemos leído contiene la siguiente información:

10
2

La función de lectura del fichero de forma convertirá el dato *10* a tipo *double* y lo grabará en la siguiente posición del vector *poseform*. Después convertirá el dato *2* a tipo *double* y lo grabará en la siguiente posición del vector *timeform*.

La información se graba en los vectores mediante la función de la clase vector *push_back()*. Esta función amplía en uno el tamaño del vector y después graba el dato introducido en la última posición, que será la recién creada. Inicialmente el tamaño de los vectores es cero, y su tamaño final será igual al número de datos que se le hayan introducido.

A continuación, si no ha llegado al final del archivo, leerá otras dos líneas. Si ha llegado, cerrará el fichero con los datos almacenados en los vectores.

Finalmente, se borrará el último elemento almacenado en ambos vectores mediante la función miembro de la clase *vector*, *pop_back()*. La función *pop_back()* elimina el último dato del vector y reduce su tamaño en uno. Se realiza este proceso porque el último dato guardado en los vectores es el *eof, end of file,* que devuelve un cero. Un cero en ambos vectores fuerza que la posición inicial sea cero, que no siempre tiene por qué serlo. Además, si en el archivo de forma ya existía una posición para el instante cero, a la hora de hacer la interpolación spline con dos puntos que comparten la misma posición del eje X, se arrojará una excepción *alglib::ap_error* y el programa finalizará. Se menciona más sobre la excepción *alglib::ap_error* en el apartado 6.2.1.2.

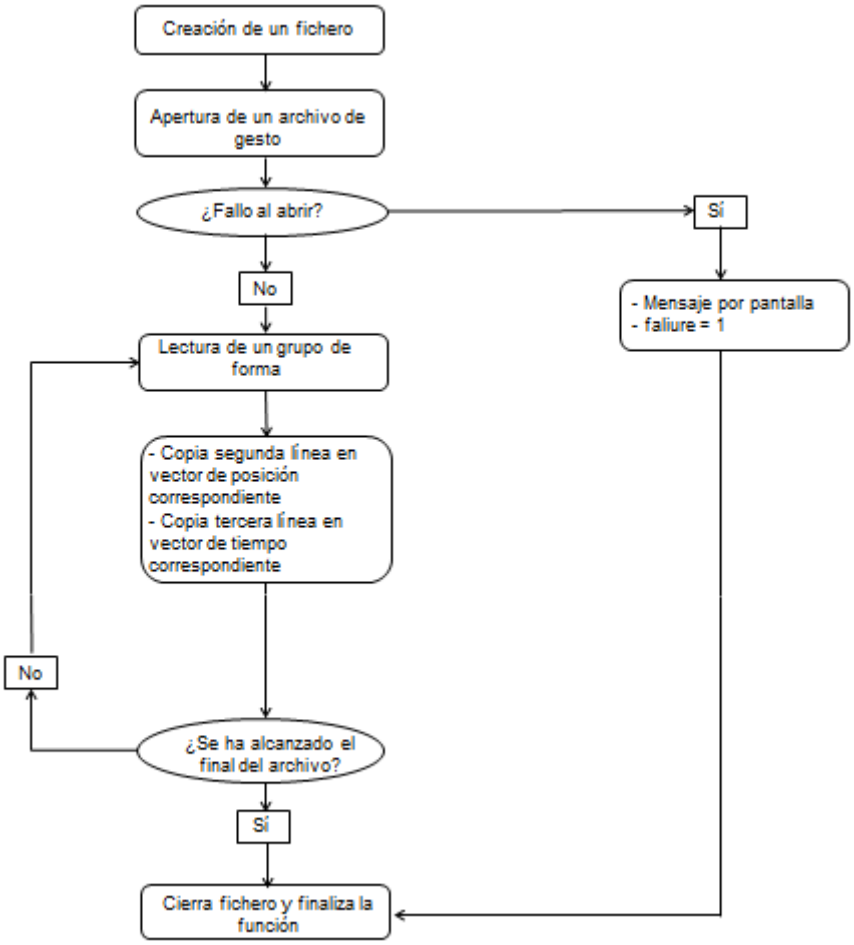


Figura 26: Diagrama de flujo de la función filetovector_field()

6.2.2.2.- Normalización de los vectores de forma. Función normalize_field()

Cómo interpretar las formas

Las formas se van a tratar como si todos sus datos estuviesen normalizados según las siguientes reglas:

- En tiempo, para todo X perteneciente al vector de tiempo de una forma, X deberá estar necesariamente comprendido en el intervalo [0,1]. Además, existe necesariamente un único X tal que X es igual a uno. Finalmente, no puede

haber dos valores X1, X2, pertenecientes al vector de tiempo de una forma que tengan el mismo valor.

$$\forall x \in Vector\ tiempo \begin{cases} x \in [0,1] \\ \exists! x, x = 1 \end{cases}$$

- En amplitud, para todo Y perteneciente al vector de posición de una forma, Y deberá estar necesariamente comprendido en el intervalo [-1, 1]. Además, existe al menos un Y tal que el valor absoluto de Y es igual a uno, es decir, que Y es o bien uno, o bien menos uno.

$$\forall y \in Vector\ posición \begin{cases} y \in [-1,1] \\ \exists y, |y| = 1 \end{cases}$$

En esta normalización, se considera que en el tiempo, el valor 0 es el inicio del gesto y el valor 1 es el final. Todos los demás valores son relativos a estos dos, por ejemplo un valor de tiempo de 0.8 significa que ha transcurrido el 80% del gesto.

En la posición, el valor -1 es la posición mínima, el valor 1 es la posición máxima y el valor 0 es el punto medio. Todos los demás valores son relativos a estos, es decir, por ejemplo el valor 0.5 es a medio camino entre el punto medio y el valor máximo, o un 75% de la amplitud total sobre la posición mínima. Al tener obligatoriamente al menos una posición con valor 1 o -1, la amplitud de la forma es la máxima posible.

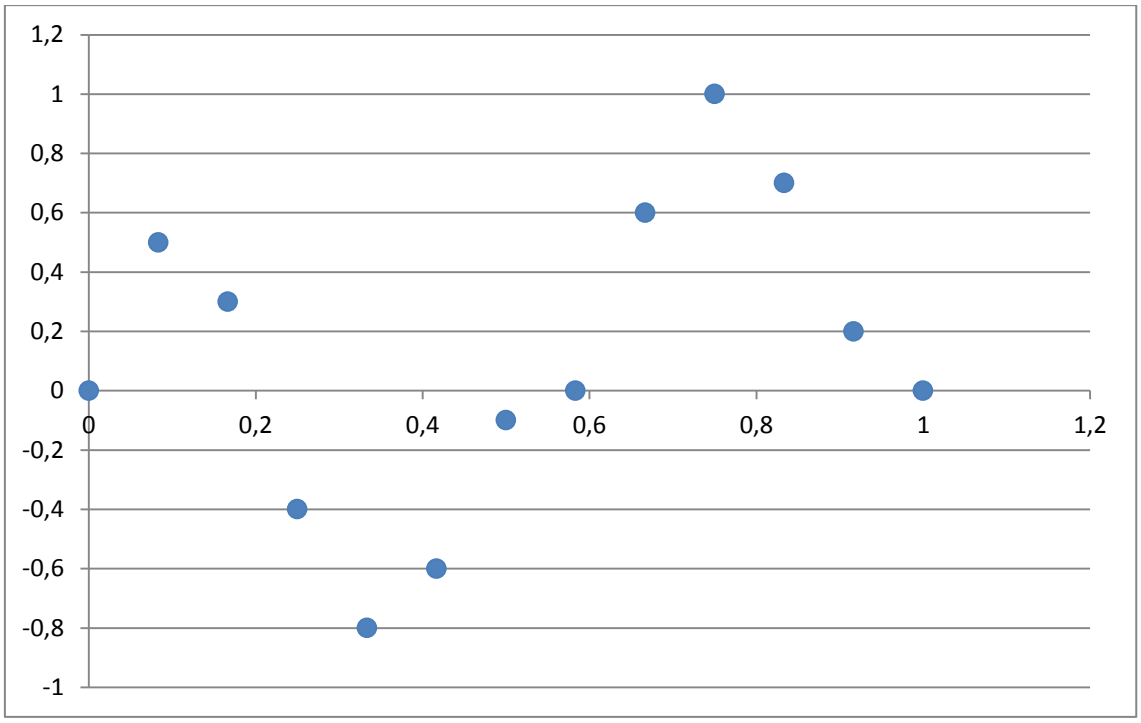


Figura 27: Ejemplo gráfico de una forma normalizada.

Sobre este estándar, se puede aplicar un bias para posición y otro para tiempo, multiplicándose todos los elementos de un vector por su bias correspondiente, manteniéndose entonces las proporciones.

Para el vector de tiempo el bias puede ser cualquier número positivo (el cero no se incluye), y el valor de ese bias será el tiempo de duración del gesto (puesto que el tiempo máximo normalizado es 1, el nuevo tiempo máximo será igual a 1*bias, es decir, el bias de tiempo).

Para el vector de posición el bias puede ser cualquier número del intervalo [-1, 1]. No puede ser un bias de valor absoluto mayor que 1, ya que la amplitud de la forma ya es la máxima posible; un bias mayor significaría que habría puntos por encima de la posición máxima o debajo de la mínima. Este bias será el porcentaje de su amplitud, es decir, si el bias es por ejemplo de 0.8, la forma se ejecutará al 80% de su amplitud, y la posición máxima (en valor absoluto) que alcance la articulación, será el 80% de su amplitud máxima, o mínima, teórica. Un bias negativo hará el movimiento simétrico (y de amplitud reducida, si el bias no es -1) con respecto al punto medio, de la forma original.

En el ejemplo de la figura siguiente se muestra el ejemplo de la figura anterior tras aplicarle un bias de 0.8 a la posición y uno de 3 al tiempo:

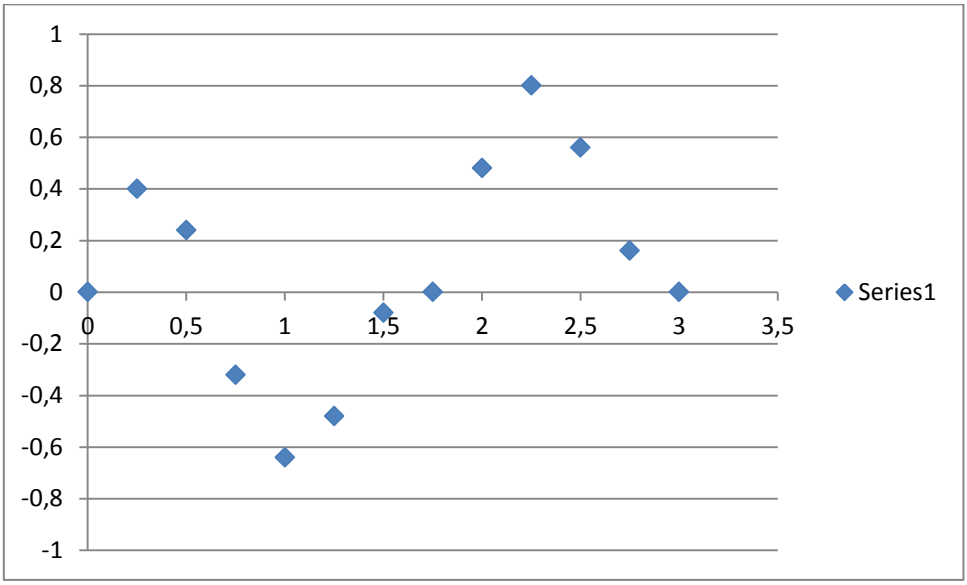


Figura 28: Ejemplo de una forma tras aplicarle un bias de posición y uno de tiempo.

Inicialmente se planteó exigir las reglas de normalización directamente en el archivo de forma. Sin embargo, para dar mayor libertad a la hora de crear y editar formas, se decidió eliminar esas restricciones. Una de las tareas de esta función será normalizar los datos para que cumplan estas reglas.

Los datos incluidos en los archivos pueden llevar cualquier amplitud y tiempo, solo se sigue aplicando la regla de que el tiempo sea positivo o cero. La amplitud de mayor valor absoluto será la máxima, que se convertirá en 1 o -1, y al dividir el resto entre ella, todos los datos quedarán proporcionales y relativos a esta. Análogamente, el tiempo máximo se convertirá en 1 y al dividir el resto entre este, todos los datos quedarán proporcionales y relativos a este.

Desarrollo de la función normalize_field()

La función *normalize_field()* comienza leyendo el valor del bias para la amplitud y para el tiempo que se va a aplicar sobre la forma normalizada, y los almacena en variables de tipo *double.* Respectivamente, *amplitude_norm* y *time_norm.*

A continuación lee todo el vector de posición de la forma y busca en él el elemento de mayor valor absoluto, que será el que indique la amplitud máxima. Guarda el valor absoluto en la variable de tipo *double, max_amplitude.* Se repite la misma operación con el vector de tiempo de la forma buscando el elemento de mayor valor. Se guarda en la variable de tipo *double, max_time.*

Por último, se normaliza la forma a través de un bucle *for*, dividiendo todos los elementos del vector de posición por el valor absoluto máximo de las amplitudes y multiplicando por el bias de posición; y dividiendo todos los elementos del vector de tiempo por el valor máximo de los tiempos y multiplicando por el bias de tiempo.

Antes de finalizar la función, se iguala la variable *gesture_time* al bias de tiempo. La variable *gesture_time* almacena el tiempo de duración total del gesto y es importante a la hora de calcular la interpolación spline.

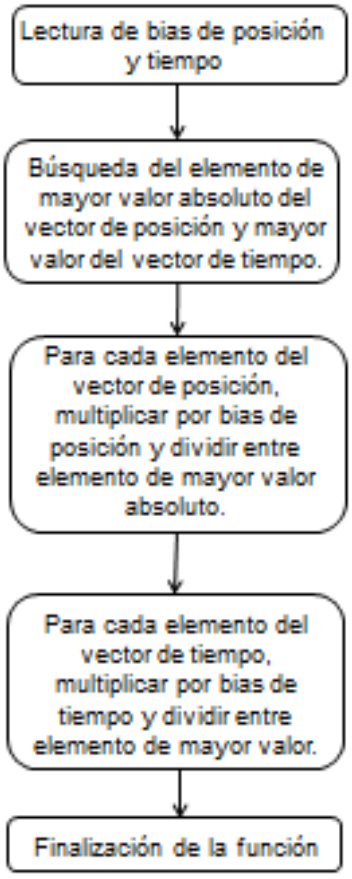


Figura 29: Diagrama de flujo de la función normalize_field().

6.2.2.3.- Conversión de la forma en un gesto. Función togesture_field()

Función que convierte una forma en un gesto que pueda ser ejecutado por una articulación del robot Maggie.

La función comienza leyendo un identificador de articulación. Este identificador indica cuál será la articulación que realice el gesto.

La función *togesture_field()* tomará los elementos del vector de posición de la forma y , en un bucle *for*, hace uno por uno la conversión de la fórmula de la siguiente figura para que los valores máximo y mínimo de la forma se ajusten a los de la articulación de forma proporcional, y se copian en el vector de posición de la articulación. En el mismo bucle *for*, se copian los elementos del vector de tiempo de la forma directamente en el vector de tiempo de la articulación. Los elementos del vector de tiempo no necesitan ser procesados.

$$Posición\ vector = \left(Posición\ forma \cdot \frac{Max_{art} - Min_{art}}{2}\right) + \frac{Max_{art} - Min_{art}}{2}$$

$$Donde:\ Max_{art} = Máxima\ amplitud\ de\ la\ articulación$$
$$Min_{art} = Mínima\ amplitud\ de\ la\ articulación$$

Figura 30: Ecuación para escalar una forma a una articulación.

Finalmente, se iguala la variable *gesture_time_(articulación)* a la variable *gesture_time* global más el antiguo valor de la propia variable *gesture_time_(articulación)* y se resetea la variable *gesture_time* a 0. La variable *gesture_time* almacena el tiempo total de duración de un gesto, y la variable *gesture_time_(articulación)* almacena el tiempo que dura el movimiento de una articulación en concreto. Esta variable es importante a la hora de calcular la interpolación spline. Existen dos motivos para hacer este paso; el primero es que la función que calcula la interpolación spline pueda usarse idéntica a como se usaba en la clase *splinegesture* sin tener que realizar ninguna modificación. La segunda es que puedan hacerse varias llamadas al trío de funciones de la clase que

leen un fichero de forma, normalizan la forma y la adaptan a una articulación para cargar formas en los vectores de distintas articulaciones antes de calcular la interpolación spline, o incluso de cargar la misma o distintas formas en una misma articulación.

6.2.2.4.- Cálculo de la spline. Función splinecalc()

El funcionamiento de esta función es idéntico al de la función del mismo nombre de la clase *splinegesture.* Para más información consultar el capítulo 6.2.1.2.

6.2.2.3.- Ejecución física del gesto. Movimiento del robot. Función gesturemovement()

El funcionamiento de esta función es prácticamente idéntico al de la función del mismo nombre de la clase *splinegesture.* La única diferencia radica en que, si va a haber un cambio de gesto, donde la función de la clase *splinegesture* llama a la función *filetovector()* para leer un fichero nuevo, la función de la clase *splineform* llama a *filetovector_field()* en su lugar. Para más información consultar el capítulo 6.2.1.3.

7.- Resultados experimentales.

Durante el proceso de desarrollo se han ido generando pruebas para comprobar que el funcionamiento teórico es el esperado en la realidad y poder continuar con la construcción del módulo. En esta categoría se incluyen pruebas con las curvas spline y pruebas con el movimiento básico de articulaciones de Maggie. En toda la fase previa a la creación definitiva de la interfaz ejecutora de gestos.

Más adelante, con el programa terminado, se usa una prueba para comprobar la funcionalidad de cada clase, y una prueba más para comprobar el funcionamiento de casos que no se corresponden al funcionamiento estándar del programa.

7.1.- Ejemplos de ejecución de gestos a partir de un fichero de gestos.

Se han desarrollado cuatro ficheros de gestos para ser desarrollados por la función splinegesture. Cada uno de esos gestos expresa una emoción distinta. Han sido desarrollados usando pocos puntos, lo que demuestra que, es posible conseguir cierta expresividad con tan solo seis grados de libertad de movimiento usando además muy pocos puntos.

El robot social Maggie tiene tan solo seis grados de libertad para expresar gestos. Esto implica que no puede aproximarse de ningún modo a la expresividad que tiene una cara humana. Por tanto, para conseguir que los gestos transmitan una emoción, se han diseñado con cierta sobre-expresividad.

Puesto que la función principal de estos gestos es comprobar que el funcionamiento del programa es adecuado, se ha tratado, además, de que intervengan el máximo número de articulaciones en cada gesto.

A continuación se detallan algunos de los gestos. Puede verse la ejecución en vídeo de cada uno en el anexo que indica cada gesto, para evaluar su efectividad de forma cualitativa. Además, se incluyen, en casi todos ellos, varias gráficas con el movimiento

que teóricamente debería seguir el robot, y que es el resultado de la interpolación spline, y, superpuesto sobre este, el movimiento que realmente sigue el robot, que se ha leído de los datos de las articulaciones mientras se efectuaba el movimiento.

7.1.1- Gesto de felicidad "Happy".

En este gesto participan todas las articulaciones. Para definir el movimiento, se han usado, como máximo, diez pares de puntos posición-tiempo por articulación en el fichero de gestos. La duración del gesto es de cinco segundos.

En el gesto se simula un baile de alegría. El vídeo de su ejecución se encuentra en el Anexo VI.

Figura 31: Happy – Brazo izquierdo.

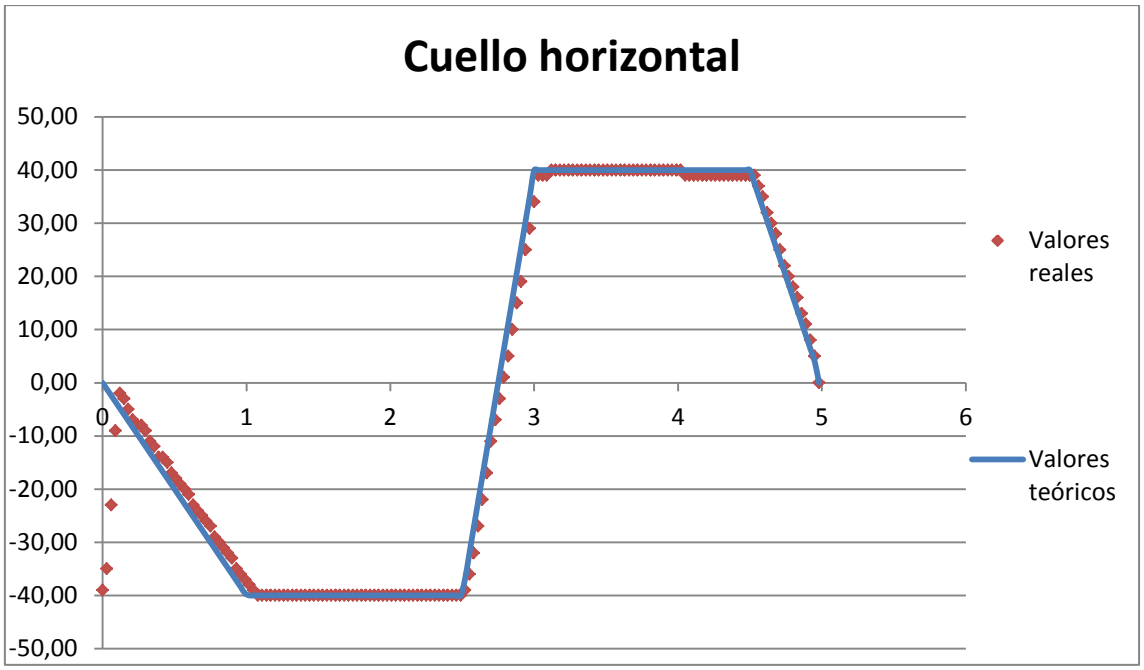


Figura 32: Happy – Cuello horizontal

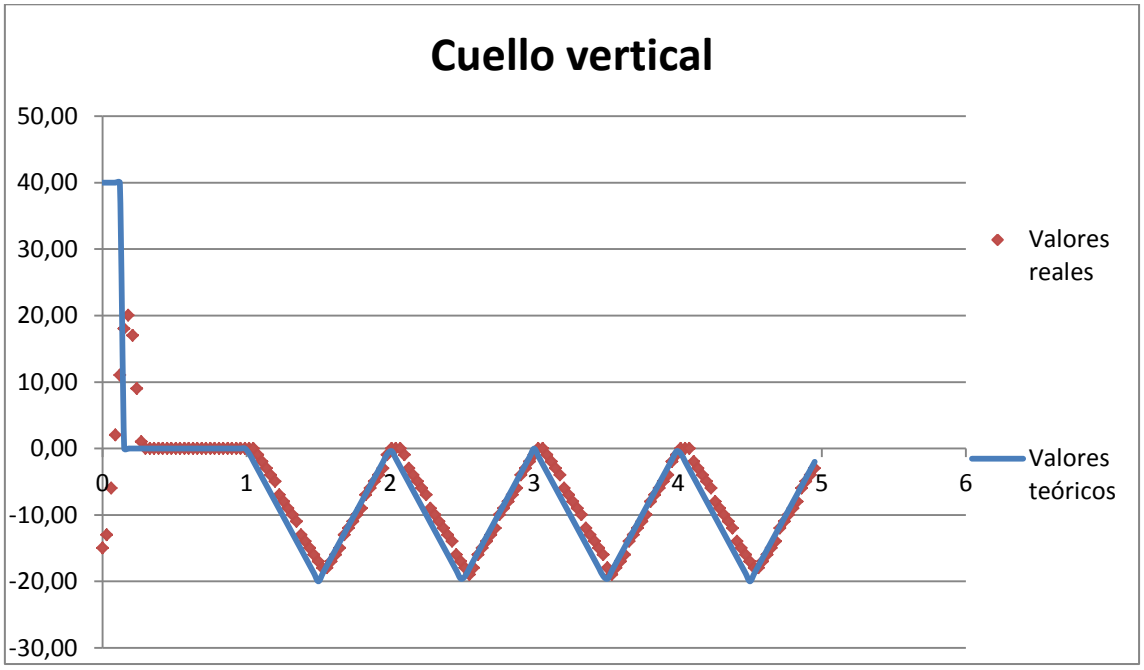


Figura 33: Happy – Cuello vertical.

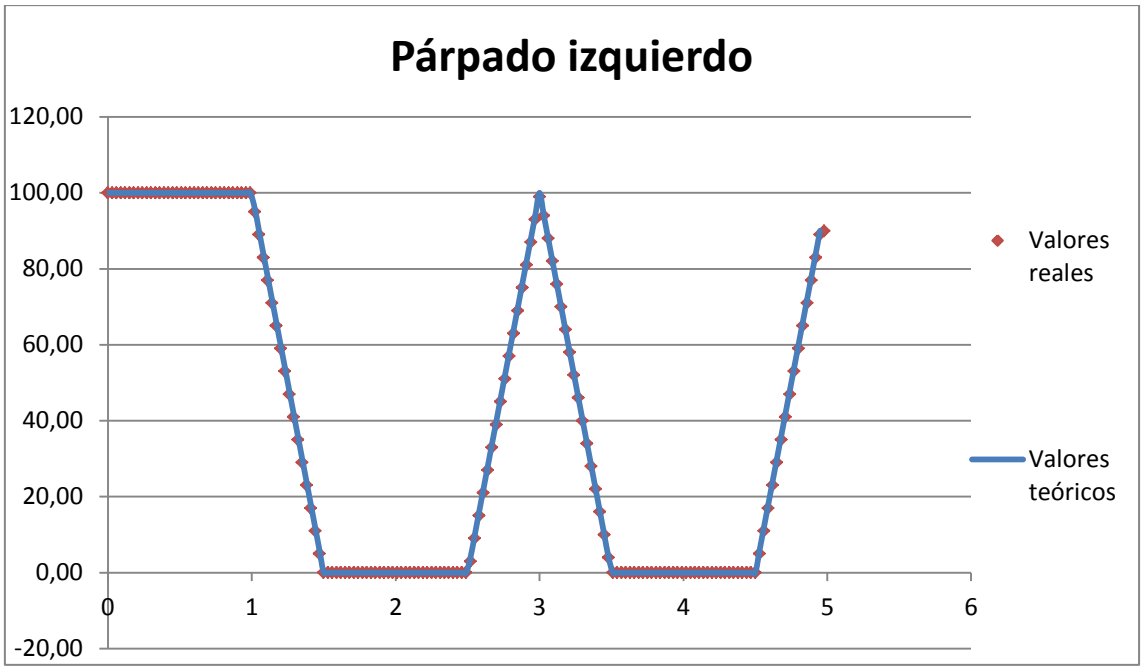


Figura 34: Happy – Párpado izquierdo

7.1.2.- Gesto de tristeza o decepción "Sad".

En este gesto participan todas las articulaciones a excepción del movimiento horizontal del cuello. Para definir el movimiento, se han usado, como máximo, diez pares de puntos posición-tiempo por articulación en el fichero de gestos. La duración del gesto es de seis segundos y medio.

El gesto comienza con el robot levantando la cabeza y los brazos hacia su interlocutor, y después bajándolos rápidamente, quedando cabizbajo, con los ojos cerrados y un pequeño balanceo de los brazos en señal de decepción. El vídeo de su ejecución se encuentra en el Anexo VII

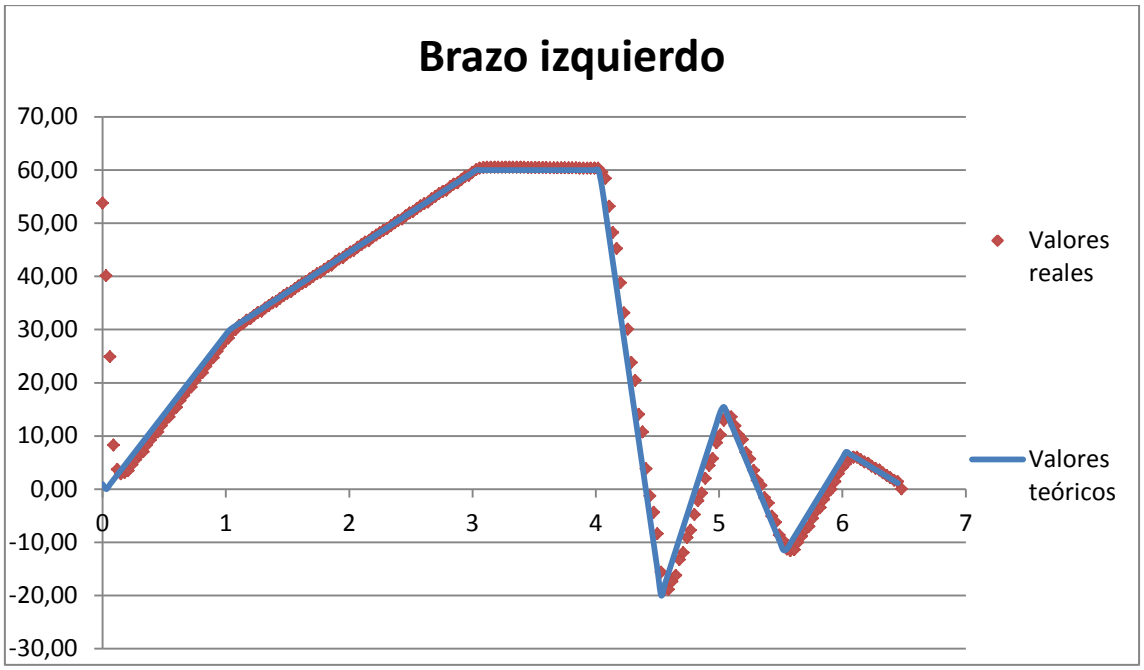


Figura 35: Sad – Brazo izquierdo

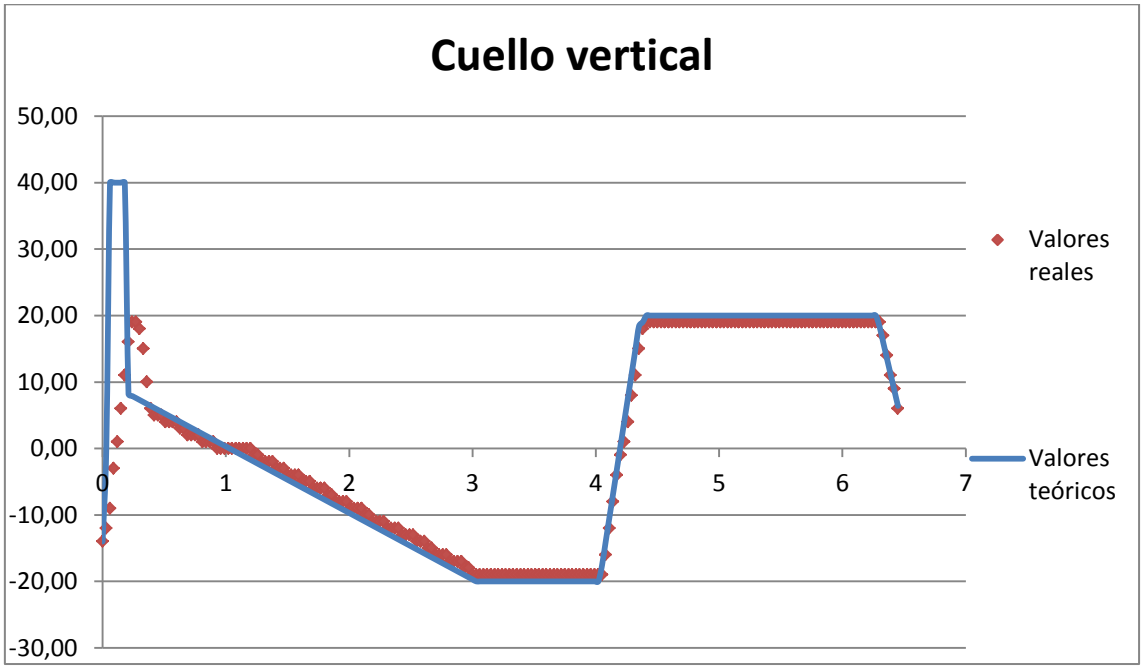


Figura 36: Sad – Cuello vertical

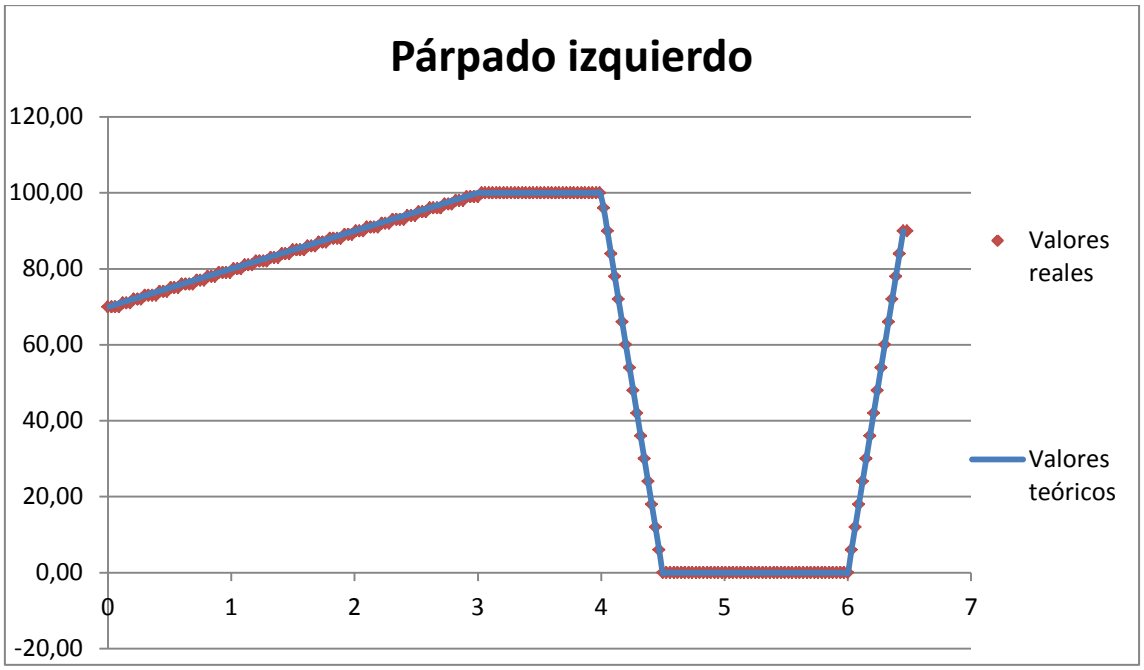


Figura 37: Sad – Brazo izquierdo.

7.1.3.- Gesto de enfado o irritación "Angry".

En este gesto participan todas las articulaciones. Para definir el movimiento, se han usado, como máximo, siete pares de puntos posición-tiempo por articulación en el fichero de gestos. La duración del gesto es de cuatro segundos.

En el gesto el robot levanta la cabeza y la agita de lado a lado con los ojos cerrados. Al mismo tiempo levanta los brazos y los mueve en el aire. El vídeo de su ejecución se encuentra en el Anexo VIII.

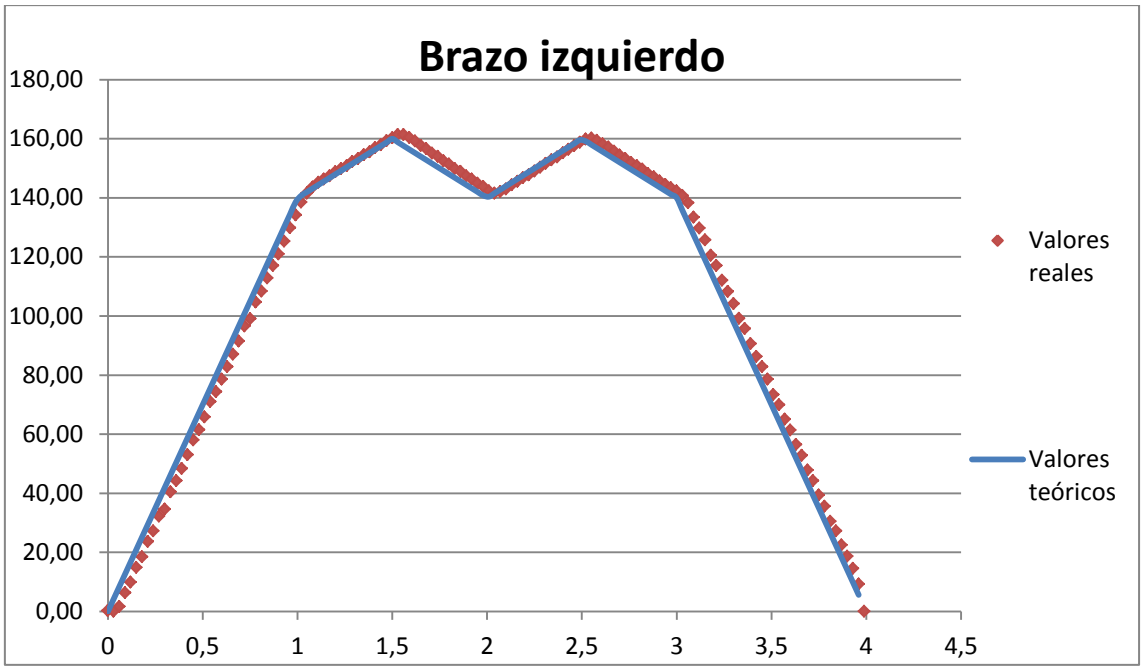


Figura 38: Angry – Brazo izquierdo.

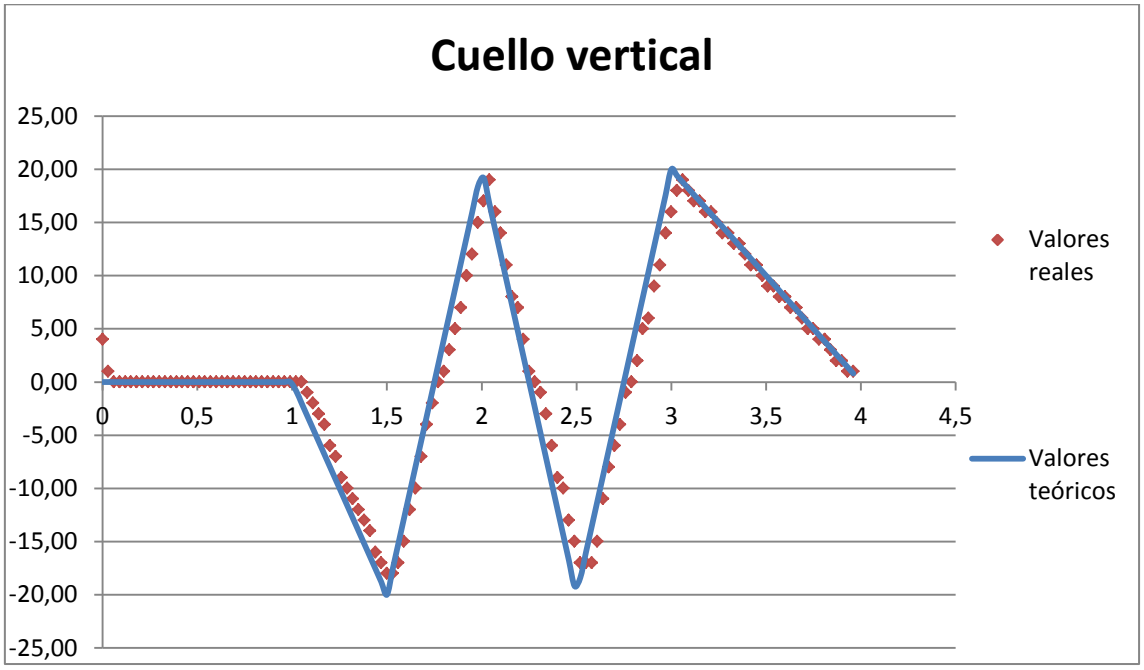


Figura 39: Angry – Cuello vertical.

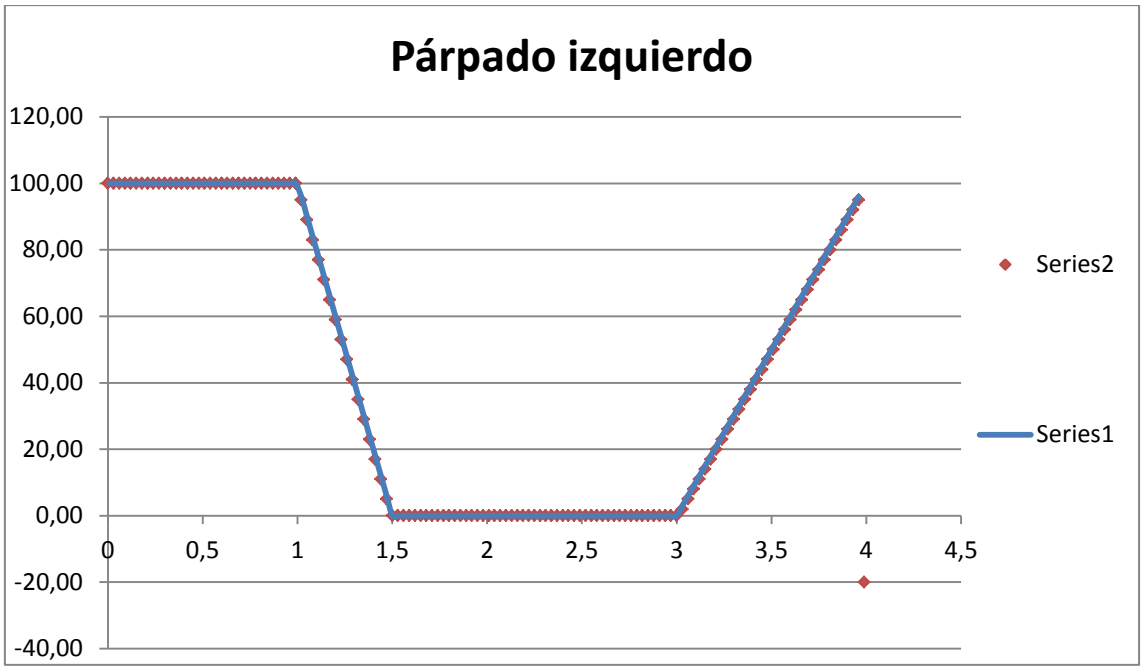


Figura 40: Angry – Párpado izquierdo.

7.1.4.- Gesto de duda o pensamiento "Thinking".

En este gesto participan los dos grados de libertad del cuello, el brazo izquierdo y el párpado derecho. Para definir el movimiento, se han usado, como máximo, siete pares de puntos posición-tiempo por articulación en el fichero de gestos. La duración del gesto es de cinco segundos.

En el gesto el robot inclina la cabeza hacia abajo y hacia su izquierda y simula rascársela con el brazo izquierdo, mientras guiña el ojo izquierdo. El vídeo de su ejecución se encuentra en el Anexo IX.

7.2.- Ejemplos de ejecución de gestos a partir de un fichero de forma.

Para realizar los test de la función splineform, simplemente se ha enviado a todas las articulaciones la misma forma, que lleva la articulación desde el cero hasta el mínimo, luego al máximo, y vuelve otra vez al cero. Se puede comprobar cómo todas las articulaciones la ejecutan igual. Además, para la articulación cuello horizontal, se

multiplica la forma primero por un bias en la posición, para reducir su amplitud, y después por un bias en el tiempo, para aumentar la duración del gesto. Se pueden ver los vídeos de la ejecución de la misma forma por distintas articulaciones en los siguientes anexos:

- Anexo X: Ejecución de la forma abstracta por el brazo izquierdo.
- Anexo XI: Ejecución de la forma abstracta por el movimiento vertical o theta del cuello.
- Anexo XII: Ejecución de la forma abstracta por el párpado izquierdo.
- Anexo XIII: Ejecución de la forma abstracta por el movimiento horizontal o phi del cuello.
- Anexo XIV: Ejecución de la forma abstracta por el movimiento horizontal o phi del cuello con un bias en la posición.
- Anexo XV: Ejecución de la forma abstracta por el movimiento horizontal o phi del cuello con un bias en el tiempo.

Nótese, que el párpado izquierdo es la única articulación en la que la posición de reposo se encuentra en un extremo (párpado completamente abierto). Por tanto, al finalizar el movimiento, donde otras articulaciones sí quedan en su posición de reposo o muy cercanas a ella, por ser el punto medio, el párpado no lo hace. Es un factor que ejemplifica muy bien el funcionamiento de las formas abstractas y debe ser tenido en cuenta a la hora de crearlas y ejecutarlas por las distintas articulaciones.

7.3.- Ejemplo de cambio de gesto durante su ejecución.

Se ha realizado un test más en el que se comprueba que la interfaz es capaz de cambiar un gesto mientras se está ejecutando. En el test, se comienza ejecutando el gesto “happy”, para después cambiarse y comenzar a ejecutar “sad” hasta el final del gesto.

El vídeo de su ejecución se encuentra en el Anexo XVI.

8.- Presupuesto.

8.1 – Desglose de horas.

Con objeto de calcular el coste, se hace un desglose de las tareas realizadas a lo largo del trabajo y el tiempo aproximado necesario para realizarlas, y de forma congruente a la planificación desarrollada en el capítulo 3.

Fase 1: Preparación:

- Investigación del estado del arte: *20 horas*.
- Conocimiento del robot Maggie: *15 horas*.
- Tutoriales de ROS: *25 horas*.
- Información spline y bibliotecas: *5 horas*.

Fase 2: Inicio del desarrollo:

- Pruebas de movimiento, splines y movimiento conjunto: *30 horas.*

Fase 3: Desarrollo:

- Desarrollo de las funciones del programa: *60 horas.*
- Pruebas del programa y correcciones: *20 horas.*

Fase 4: Ampliación y finalización del trabajo.

- Desarrollo de la ampliación: *5 horas.*
- Pruebas finales: *5 horas.*
- Optimización y comentario del código: *5 horas.*
- Redacción de la memoria: *90 horas.*

FASES	HORAS
Fase 1. Preparación.	65
Fase 2. Inicio del desarrollo.	30
Fase 3. Desarrollo.	80
Fase 4. Ampliación y finalización del trabajo	105
Total	**280**

Figura 41: Desglose de horas de la realización del trabajo.

8.2.- Costes materiales.

Los costes materiales han sido un ordenador de trabajo y el alquiler del robot Maggie. El ordenador se considera con un periodo de amortización de 3 años. El alquiler del robot Maggie se estima a partir de los costes de compra de otros robots comerciales, con un periodo de amortización de 5 años [17] [18]. Se calcula teniendo en cuenta el tiempo del proyecto del apartado anterior.

CONCEPTO	PRECIO (€)
Ordenador de trabajo.	150
Alquiler de Maggie	1200
Total	**1350**

Figura 42: Costes materiales.

8.3.- Costes de personal.

En la realización del presente proyecto ha sido necesaria la presencia de un jefe de proyecto y un ingeniero.

PERSONAL	HORAS	PRECIO/HORA	IMPORTE (€)
Jefe de proyecto	22	90	1980
Ingeniero	258	60	15480
Total	**280**		**17460**

Figura 43: Costes de personal.

8.4.- Coste final.

CONCEPTO	PRECIO (€)
Costes materiales.	1350
Costes de personal.	17460
Costes indirectos (20%)	3762
Subtotal	22572
IVA (18%)	4062,96
Total	**26634,96**

Figura 44: Coste final del proyecto.

El coste total del proyecto es de *veintiséis mil seiscientos treinta y cuatro euros con noventa y seis céntimos.*

9.- Trabajos futuros.

Este trabajo es un comienzo básico. Permite la ejecución de gestos, que es un paso importante en la interacción de humanos con robots, y un componente fundamental en los robots sociales.

Sin embargo, es posible desarrollar mejoras a estas APIs, e integrarlas en otras aplicaciones futuras.

9.1.- Futuros usos de la aplicación.

Las nuevas aplicaciones en el Robotics Lab Uc3m han ido, durante el último año, abandonando la arquitectura AD y centrándose en la arquitectura ROS. Además, a partir de noviembre de 2012, se empiezan a utilizar las últimas versiones de Ubuntu y ROS. El robot Maggie es incompatible con estas actualizaciones, y, deja de poder utilizar o actualizar las partes de su arquitectura que tengan módulos de ROS.

Sin embargo, aunque no vayan a utilizarse en el propio Robot Maggie, es posible utilizar las aplicaciones relacionadas con este trabajo en algunos de los futuros proyectos del laboratorio.

Téngase en cuenta que los dos proyectos que aquí se exponen no están todavía confirmados, los modelos que se citan son únicamente posibilidades, y es posible que el modelo final que se utilice no sea compatible con ellos.

9.1.1.- Proyecto Alzheimer.

En el proyecto Alzheimer, el objetivo principal es mejorar la calidad de vida de los pacientes y proveer de un apoyo a los cuidadores mediante el uso de un robot personal.

En este proyecto se ha trabajado de la mano con la “Fundación Alzheimer España” para ayudar a definir temas como el aspecto físico del robot, la funcionalidad y el tipo de

interacciones que llevará a cabo, y desarrollar una lista de posibles escenarios que puedan ser llevados a cabo por un robot social.

Las principales tareas en las que el robot puede ayudar al paciente son:

- Estimulación, mediante ejercicios de memoria o terapias musicales.

- Entretenimiento, mediante contar historias, reproducir la televisión o la radio, escuchar al paciente o enseñarle fotos. Todo con el objetivo de ayudar a que el paciente no se sienta solo y crear vínculos afectivos con él.

- Asistencia personal, mediante la ayuda en tareas diarias y actividades básicas.

- Seguridad, mediante una función de vigilancia para avisar al asistente en caso de comportamiento inusual.

Aunque la apariencia física del robot aún no está confirmada, uno de los prototipos sobre los que se está trabajando es muy similar al robot Maggie, aunque de menor tamaño.

Figura 45: Posible prototipo del robot Alzheimer.

Si el robot fuese como este, o uno con las mismas articulaciones, podría reutilizarse todo el código de las dos clases de este trabajo. Si el robot es distinto, habría que adaptarlo a las posibles diferencias.

9.1.2.- Proyecto MOnarCH.

MOnarCH es un proyecto europeo cuyo objetivo es usar robots para interactuar con niños, personal y visitantes, en actividades de entretenimiento educativo en el ala de pediatría del Instituto Portugués de Oncología de Lisboa (IPOL). [11] [18]

El proyecto está en desarrollo y aún no se tienen prototipos del tipo de robots que se usarán. Como en el caso anterior, si es de forma similar a Maggie, podrá reutilizarse todo el código de ambas clases. Si es distinto, podría adaptarse a las posibles diferencias. Hay que tener en cuenta además que la clase *splineform* de este trabajo utiliza formas, que son universales y pueden ser utilizadas por cualquier robot, siempre y cuando el procesado de esas formas se haga específico a ese robot.

9.2.- Mejoras en los gestos.

Los gestos que se han creado para este programa son de prueba y muy básicos. A pesar de eso, demuestran cierta expresividad. Sin embargo, en aplicaciones futuras sería muy interesante desarrollar gestos a partir de un estudio previo.

Para un conjunto de gestos realmente válido para la interacción con humanos, sería interesante, en primer lugar, consultar bibliografía relacionada con el lenguaje corporal y la psicología. Y combinarlo con un estudio demográfico. No todos los grupos de personas comprenden o admiten del mismo modo los diferentes tipos de lenguaje corporal.

A partir de ese estudio, se podría crear un conjunto de gestos más amplio y robusto que el actual, y que pueda utilizarse en un mayor abanico de situaciones, o especializarse en un grupo concreto si el robot va a encontrarse en un ámbito determinado; por ejemplo, en alguno de los dos escenarios que se proponen en el apartado anterior.

Por último, realizar pruebas con distintos sujetos para comprobar que el conjunto de gestos escogido e implementado obtiene los resultados deseados.

9.3- Ampliaciones en el programa.

Sobre el trabajo que aquí se presenta, aún es posible crear varias ampliaciones que mejoren el rendimiento u ofrezcan características extra a las que ya existen.

9.3.1- Implementación en la arquitectura ROS.

Breve descripción de ROS.

Robot Operating System, más conocido como ROS, es un framework para desarrollo de sistemas y software relacionados con la robótica. Puede funcionar de forma similar a un sistema operativo para un conjunto de máquinas. Fue desarrollado en 2007 por el departamento de inteligencia artificial de la universidad de Stanford. [19]

ROS ofrece servicios de un sistema operativo estándar, como abstracción de hardware, control de dispositivos a bajo nivel, implementación de funcionalidades comunes, mensajería entre procesos y manejo de paquetes.

ROS está basado en una arquitectura de grafos. Los programas, aplicaciones, habilidades, etcétera corresponden a los nodos del grafo. Los topics corresponden a las aristas (orientadas) del grafo. Cada nodo puede suscribirse a tantos topics como desee, y puede también publicar mensajes en topics. Cuando un nodo publica un mensaje en un topic, todos los nodos suscritos en ese topic reciben el mensaje. Esta es la forma básica de comunicación entre los nodos de la arquitectura; pero también pueden hacerlo, por ejemplo, a través de servicios, que son una forma de comunicación directa nodo a nodo a través de una petición (request) y una respuesta (response). En la siguiente figura se muestra un ejemplo de comunicación ente nodos (óvalos) a través de topics (flechas). El nodo que publica en un topic es el del origen de la flecha, mientras que el que está suscrito al topic se encuentra en el final de la flecha.

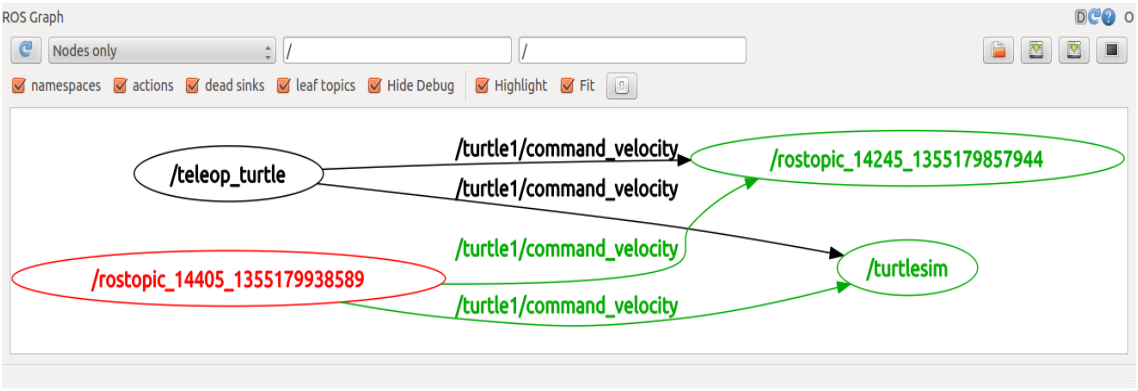


Figura 46: Ejemplo gráfico de funcionamiento de ROS.

Además del sistema operativo descrito arriba, ROS incluye otro "lado" llamado ros-pkg, que consiste en una suite de paquetes contribución de los usuarios, organizados en sets llamados stacks. Que implementan diversas funciones como percepción, planificación, simulación, etcétera.

La arquitectura ROS es de software libre bajo licencia BSD. Los stacks de ros-pkg pueden estar protegidos por diferentes licencias según el usuario que realice la contribución.

Maggie y ROS.

Se ha explicado en el capítulo 9.1. que el robot Maggie es actualmente incompatible con ROS. Sin embargo, se ha desarrollado este trabajo pensando en la posibilidad de que en un futuro pudiese integrarse con ROS. Para ello, principalmente habría que sustituir las entradas por teclado por mensajes a través de los topics de ROS. También existen variables, como el flag que permite modificar un gesto durante su ejecución, que están pensadas para que en un futuro puedan ser operadas con mensajes de ROS.

Es posible que las aplicaciones de este trabajo se incluyan en nuevos robots, compatibles con la plataforma ROS, como se ha comentado más arriba. En este caso, sería no solo interesante, sino probablemente necesario, el adaptar este código a ROS.

9.3.2.- Aprendizaje de gestos.

Se plantea también la posibilidad de un aprendizaje de gestos del robot.

9. Trabajos futuros.

Un robot podría aprender nuevos gestos a través de la adquisición de formas que más adelante combinaría entre ellas o no y las convertiría en gestos que más adelante podría ejecutar.

Existen varias posibilidades para que un robot aprenda gestos.

- A través de comandos de voz. Ya existe en el Robotics Lab de la Uc3m una herramienta para comandar por voz, y se ha probado en el robot Maggie con éxito. Sería necesario entonces, a partir de ese trabajo, crear una API que fuese capaz de reconocer los movimientos que se realizan y convertirlos en una forma.

- A través del tacto. Maggie posee sensores de tacto, por tanto sería posible crear una API donde un operador le moviese las articulaciones y este movimiento se fuese guardando en archivos de forma.

- A través de una cámara. Existen ya algunas aplicaciones con la cámara Kinect de Xbox 360 de reconocimiento del cuerpo y movimientos de un ser humano. Sería necesario adaptar esos movimientos a los posibles movimientos de Maggie, más limitados por ser más simples sus articulaciones, y convertir esto en un fichero de forma.

- A través de otros robots. Las formas están planteadas para poder ser usadas por cualquier robot, por tanto, es viable que puedan enseñarse entre ellos a realizar nuevos movimientos; bien a través de alguno de los métodos que hemos descrito aquí, bien enviando directamente el archivo, por ejemplo, a través de la red WiFi.

10.- Conclusiones.

Se han conseguido desarrollar dos interfaces para el robot social Maggie. La primera de ellas es capaz, de leer los datos de un fichero de gesto, usar esos datos para interpolar una curva spline, tomar puntos de esa interpolación e interpretarlos como posiciones de las articulaciones del robot para generar movimiento en Maggie.

La segunda de ellas, es capaz de leer los datos de un fichero de forma abstracta, válido para cualquier articulación, normalizar la forma y cambiar su amplitud total o su duración, asignarla a una articulación, realizar la misma operación de interpolación que la primera interfaz y generar movimiento en Maggie.

Además, se han desarrollado cuatro gestos que demuestran, por un lado, que la interfaz funciona perfectamente, y por otro, que usando esta interfaz es posible conseguir bastante expresividad en el robot a partir de pocos puntos, teniendo además como base de trabajo muy pocos grados de libertad.

Uno de los objetivos planteados al introducir las curvas spline para el movimiento era conseguir un control más preciso de las articulaciones. Al enviarles una nueva posición cada 30 milisegundos, se evitan posibles desvíos en trayectorias más largas. A la vista de los resultados experimentales, podemos concluir que se ha conseguido en gran medida. Especialmente si consideramos las gráficas relativas al párpado, que, al moverse mediante un servomotor con placa de control en vez de un motor de corriente continua como el resto de articulaciones, es mucho más preciso.

Además, se ha conseguido mover todas las articulaciones del cuerpo de Maggie a partir de un único fichero de forma abstracta. Es decir, que enviando los mismos datos de partida cualquiera de las articulaciones de Maggie es capaz de interpretarlos como movimientos propios. Es posible además, modificar la amplitud de una forma abstracta, o la duración de la misma antes de llegar a ejecutarse. De este modo, una misma forma abstracta puede utilizarse en distintas situaciones y contextos, haciendo más o menos énfasis según sea necesario.

Finalmente, se ha conseguido realizar la ampliación propuesta al principio, que consiste en incluir la posibilidad de que el gesto se cambie durante la ejecución del mismo.

Este trabajo representa el colofón a varios años del estudio de ingeniería. Es el resultado del aprendizaje en las distintas asignaturas, y muchos de los conocimientos adquiridos durante la carrera se reflejan en él.

La robótica personal está cada vez más presente, y su futura integración mayoritaria en nuestro día a día parece cada vez más evidente. La idea de un robot social es que lo aceptemos como parte de nuestro entorno, que podamos comunicarnos con él con cierta dosis de afecto, no como si fuese un electrodoméstico más.

El ser humano, como ser social, está abierto a la empatía. Llegará el día en que sea capaz de empatizar con un robot.

11.- Bibliografía.

[1] Merhabian, Albert (2012). *Nonverbal Comunication*. University of California, Los Ángeles

[2] Aldebarán Robotics: http://www.aldebaran-robotics.com/en/

[3] Wikipedia – NAO: http://es.wikipedia.org/wiki/Nao_(robot)

[4] JTSC: http://home.comcast.net/~jtechsc/

[5] Asimo Honda: http://asimo.honda.com/

[6] Wikipedia – Asimo: http://en.wikipedia.org/wiki/ASIMO

[7] Wikipedia – QRIO: http://en.wikipedia.org/wiki/QRIO

[8] Wikipedia – Kismet: http://en.wikipedia.org/wiki/Kismet_(robot)

[9] Gorostiza, Javier (2010). *Programación natural de un robot mediante diálogos*. Universidad Carlos III de Madrid, España.

[10] Wikipedia – Leonardo: http://en.wikipedia.org/wiki/Leonardo_(robot)

[11] Robotics Lab Uc3m: http://roboticslab.uc3m.es/mediawiki/index.php/Main_Page

[12] Barber, Ramón (2001). *Desarrollo de una arquitectura para robots móviles autónomos. Aplicación en un sistema de navegación topológica*. Universidad Carlos III de Madrid, España.

[13] Alglib documentation: http://www.alglib.net/interpolation/spline3.php

[14] Wikipedia – Spline: http://en.wikipedia.org/wiki/Spline_(mathematics)

[15] Dreamincode: http://www.dreamincode.net/forums/topic/33631-c-vector-tutorial/

[16] CMake: http://www.cmake.org/

[17] Robot Shop: http://www.robotshop.com/store

[18] RoboSavy: http://robosavvy.com/site/

[19] Proyecto Monarch: http://monarch-fp7.eu/

[20] ROS: http://www.ros.org/wiki/